Titre: Title:	Intégration d'heuristiques d'incitation à la programmation par contraintes
Auteur: Author:	Abdallah Oumha
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Oumha, A. (2006). Intégration d'heuristiques d'incitation à la programmation par contraintes [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7730/

UNIVERSITÉ DE MONTRÉAL

INTÉGRATION D'HEURISTIQUES D'INCITATION À LA PROGRAMMATION PAR CONTRAINTES

ABDALLAH OUMHA
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2006

Your file Votre référence
ISBN: 978-0-494-17964-2
Our file Notre référence
ISBN: 978-0-494-17964-2

Ce mémoire intitulé:

INTÉGRATION D'HEURISTIQUES D'INCITATION
À LA PROGRAMMATION PAR CONTRAINTES

présenté par : ABDALLAH OUMHA
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. GAGNON Michel, Ph. D., président
M. PESANT Gilles, Ph. D., membre et directeur de recherche
M. GALINIER Philippe, Doct., membre et codirecteur de recherche
M. ROUSSEAU Louis-Martin, Ph. D., membre

Je dédie ce travail à mes parents, Ahmed Oumha et Khira Harbach, ainsi qu'à ma compagne Naâïma pour leur soutien et leur prières,.

Remerciements

Je tiens tout d'abord à remercier mes directeurs Gilles Pesant et Philippe Galinier pour leur aide, leurs conseils, leur soutien autant moral que financier sans oublier le considérable travail de correction qu'ils ont effectué. Je tiens à remercier particulièrement Gilles Pesant pour son aide concernant la programmation par contraintes et ses encouragements qui m'ont permis de persévérer. Je remercie Philippe Galinier particulièrement pour sa présence et son aide en heuristiques.

Mes remerciements sont aussi adressés à Louis Martin Rousseau pour son aide en programmation par contraintes et ses encouragements. Je remercie aussi les membres du laboratoire de programmation par contraintes « Quosséça » Alessandro Zanarini et Simon Boivin pour leur soutien technique occasionnel et leurs encouragements. Sans oublier les membres du Centre de recherche sur les transports (CRT) pour leur soutien technique, particulièrement les analystes informaticiens.

Des remerciements distincts sont destinés aux membres de ma famille qui n'ont pas cessé de m'encourager des deux côtés de l'Atlantique. Spécialement mes parents, Ahmed Oumha et Khira Harbach, et ma femme Naâïma.

Résumé

Ce mémoire traite de l'intégration d'heuristiques d'incitation robustes et génériques à la stratégie de recherche d'une application en programmation par contraintes. Nous visons l'amélioration de la robustesse d'une application de confection d'horaires en milieu hospitalier nommée Hibiscus.

Nous décrivons une méthode flexible et générale de conception d'heuristiques d'incitation. Chaque heuristique est associée à une contrainte du problème et elle attribue, selon certains critères, une note aux assignations. On entend par assignations les couples (variable, valeur) possibles selon l'état de la recherche. Une note cumulée par assignation est obtenue par pondération des notes attribuées par les heuristiques d'incitation selon leur poids. La stratégie de recherche choisit l'assignation ayant la plus grande note cumulée.

Nous avons implémenté deux heuristiques d'incitation. La première concerne les contraintes de type Cardinalité alors que la deuxième concerne les contraintes de type Cardinalité pondéré. Les tests de validation effectués ont montré une amélioration considérable de la robustesse d'Hibiscus.

Mots Clefs : heuristiques d'incitation, heuristiques de choix de valeur, confection d'horaire, milieu hospitalier, programmation par contraintes.

Abstract

This master's thesis presents the integration of generic and robust incentive heuristic to the search strategy of an application in constraint programming. Our main objective is the improvement of a staff scheduling application in health care named Hibiscus.

We describe a flexible and general design method for incentive heuristics. Each heuristic is associated with one constraint and it assigns a score to each possible assignment according to certain criteria. Assignments is all (variable, value) couple possible regarding the search state. The weighting of the notes allotted by the incentive heuristics, according to the incentive heuristics weight, allows the computation of a cumulative score for each assignment. The search strategy selects the assignment having the greatest cumulative score.

We implemented two incentive heuristics. The first one relates to the Cardinality constraint and the second relates to the balanced Cardinality constraint. The validation tests showed a considerable improvement of Hibiscus's robustness.

Key Words : incentive heuristic, value choice heuristic, staff scheduling, health care, constraint programming.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et abréviations

CSP	**C**onstraint **S**atisfaction **P**roblem (Problèmes de Satisfaction des Contraintes).
CSOP	**C**onstraint **S**atisfaction and **O**ptimization **P**roblem (Problèmes d'Optimisation et de Satisfaction des Contraintes).
PPC	Constraint Programming (**P**rogrammation **P**ar **C**ontraintes).
MAC	**M**aintaining **A**rc **C**onsistency (Maintien de la cohérence d'arc).
CPU	**C**entral **P**rocessing **U**nit (Unité de traitement central).
DFS	**D**epth **F**irst **S**earch (Recherche en profondeur d'abord).
LDS	**L**imited **D**iscrepancy **S**earch (Recherche avec divergence limitée).
DDS	**D**epth-bounded **D**iscrepancy **S**earch (**LDS** à profondeur croissante)
FF	**F**ail **F**irst (échec d'abord)
SF	**S**ucceed **F**irst (Succès d'abord)
SLC	**S**tatic **l**east-**c**onflicts.
SMP	**S**tatic **m**ax-**p**romise.
LVO	look-ahead **v**alue **o**rdering.
ABL	**A**verage **B**ranch **L**ength (longueur moyenne des branches).
DEM	**Dem**and (La demande).
WOR	**Wor**k load (charge de travail).
AVA	**Ava**ilability (Disponibilité).
ERG	**Erg**onomic (Ergonomie).
FAI	**Fai**rness (Équité).
WEF	**W**eek **E**nd **F**irst (Les fins de semaines d'abord).
MNoSF	**Min** Numbe **O**f **S**olution **F**ound (Nombre Minimum de solutions trouvées).
NBN	**N**urse **B**y **N**urse (Infirmière par Infirmière).
DBD	**D**ay **B**y **D**ay (Jour par jour).
MAC	**M**aintaining **A**rc **C**onsistency.

Liste des annexes

Introduction

La résolution des problèmes NP-difficiles constitue un défi intéressant vu leur difficulté et leur abondance dans plusieurs domaines, par exemple en intelligence artificielle. Un problème NP-difficile est caractérisé entre autres par son énorme espace de recherche (espace de recherche exponentiel par rapport à la taille du problème). La résolution des problèmes NP-difficiles fait souvent appel aux heuristiques parce que les algorithmes exacts et efficaces tardent à apparaître. D'où notre intérêt pour l'intégration des heuristiques à la programmation par contraintes.

Notre objectif principal est de proposer un cadre de recherche de solutions robuste et modulaire aux problèmes NP-difficiles qui est adapté à la PPC (programmation par contraintes). Plus précisément, nous proposons des heuristiques d'incitation qui ne sont liées ni au modèle ni aux contraintes du problème étudié. Les heuristiques d'incitations proposées permettent de diriger mieux le processus de recherche de solution.

En génie logiciel, la robustesse et la modularité sont deux caractéristiques importantes d'un logiciel. La conception d'un logiciel concurrentiel, convivial et évolutif passe impérativement par la considération desdites caractéristiques. Un logiciel est dit robuste s'il ne plante pas et s'il fonctionne bien [37]. En général, un produit est dit robuste si sa réponse est peu modifiée par des perturbations. On entend par perturbations les changements des conditions de fonctionnement. Un produit optimisé, mais qui ne fonctionne que dans des conditions particulières ne sera pas robuste. Dans notre cas, les

perturbations peuvent être obtenues par l'ajout, le retrait, l'activation et la désactivation des contraintes ou simplement le changement de jeu de données en entrée.

La modularité vise le maintien de la flexibilité et l'évolutivité du système. Un système est dit modulaire s'il est constitué d'entités fortement indépendantes [29] et ces entités sont constituées d'un ensemble de programmes et structures de données fortement cohérent et consistant. La programmation modulaire est à l'origine de la programmation orientée objet. La modularité permet la séparation entre l'implémentation et l'interface, la réutilisation des modules, la réduction du temps de conception, la conception parallèle (plusieurs personnes travailler en même temps sur un même projet), la réduction du temps de maintenance et la génération de logiciel évolutive (logiciels aux quels on peut ajouter des fonctionnalités facilement, qui évolués à la demande).

Le mot *heuristique* provient de la même racine grecque que la fameuse expression *Eureka* (j'ai trouvé) prononcée par Archimède. Plusieurs définitions peuvent être, et sont, utilisée(s) pour décrire ce qui est une heuristique. En informatique, une heuristique est une méthode de résolution des problèmes qui n'est pas basée sur un modèle formel et qui n'aboutit pas forcément à une solution. Les heuristiques sont, à la différence des algorithmes exacts, tirées de l'expérience et d'analogies, plutôt que d'une analyse scientifique trop complexe qui doit recenser le maximum d'éléments. Cependant, le choix de la bonne heuristique nécessite une connaissance approfondie des propriétés du problème que l'on s'apprête à résoudre.

Les heuristiques que nous proposons sont centrées sur les contraintes et collaborent ensemble pour faire un bon choix de la valeur à instancier. À certaines

contraintes '*C*' du problème, on associe une instanciation d'heuristique '*h*'; '*h*' est dite heuristique liée à '*C*'. Chaque heuristique '*h*' possède un poids (une pondération) qui est proportionnel à son importance dans le problème étudier. L'heuristique '*h*' accorde une note à chaque assignation, couple (variable, valeur), selon son impact sur la satisfaction de '*C*' vis-à-vis l'état courant de la recherche. Un score cumulé pour chaque assignation est obtenu en effectuant une pondération des notes attribuées par les heuristiques '*h*' selon leur poids. L'assignation choisie est celle ayant le meilleur score cumulé. Ainsi, une collaboration entre toutes les heuristiques '*h*' est assurée afin de mieux guider la recherche.

L'application en programmation par contraintes Hibiscus, dédiée à la confection d'horaires non cycliques pour les infirmières et les médecins et développée dans le cadre de travaux de maîtrise [2], affiche une performance dépendant grandement du jeu de données utilisé. Rappelons que les problèmes de génération d'horaire sont des problèmes NP-difficiles.

Notre objectif est l'amélioration de la robustesse d'Hibiscus et le maintien de sa modularité. L'amélioration de la robustesse vise le maintien du pourcentage de réussite à un bon niveau indépendamment du problème en entrée (jeu de données). Pour atteindre notre objectif, nous proposons des heuristiques d'incitation, robustes et modulaires, pour guider la recherche. Plus précisément, nous proposons des heuristiques de choix de valeur vu l'impact important de ce choix sur les performances d'une application en programmation par contraintes.

Organisation du mémoire

Dans le premier chapitre, nous présentons les problèmes de satisfaction des contraintes (CSP : *Constraint Satisfaction Problem*) et les composantes clés de la programmation par contraintes. Un tour d'horizon de la littérature au sujet des heuristiques de choix de valeur utilisées dans les stratégies de recherche clora ce premier chapitre.

Dans le second chapitre, nous décrivons le problème de validation choisi, soit la génération des horaires, puis l'application de validation Hibiscus.

Dans le chapitre trois, nous présentons le principe général des heuristiques d'incitation proposées. Puis, nous abordons le premier type de contrainte traité (la Cardinalité) ainsi que le deuxième type de contrainte traité (la Cardinalité pondérée). Ensuite, nous présentons les contraintes de validation retenues (Demande et Équilibrage) ainsi que les applications des heuristiques d'incitation à celles-ci.

Dans le chapitre quatre, nous décrivons les jeux de données utilisés, le plan d'expérimentation retenu et les résultats obtenus. Une discussion des résultats obtenus clôt ce chapitre.

Dans le chapitre cinq, nous présentons les conclusions et les limites de ce travail de recherche. Puis, nous abordons les perspectives de recherche et d'amélioration proposées dans le cadre de l'intégration des heuristiques d'incitation aux stratégies de recherche.

Chapitre 1 : Programmation par contraintes

Dans ce chapitre, nous présentons les problèmes de satisfaction des contraintes (CSP : *Constraint Satisfaction Problem*). Puis, nous discutons des composantes clés de la programmation par contraintes. Ensuite, nous détaillons une composante principale de ce paradigme, soit la stratégie de recherche. On clora ce chapitre par un tour d'horizon de la littérature au sujet des heuristiques de choix de valeur utilisées dans les stratégies de recherche.

1.1 Problème de satisfaction de contraintes (CSP)

Un CSP permet de présenter un problème du monde réel sous forme d'ensemble de contraintes et de variables. Les contraintes forment un concept courant dans nos vies quotidiennes. En effet, l'heure du réveil, le trafic, les heures de bureau, etc., s'expriment facilement sous forme de contraintes. Parmi les types des contraintes, on trouve les contraintes binaires, arithmétiques, linéaires, etc. Formellement, de nombreux problèmes réels s'expriment sous forme d'ensemble de contraintes et d'ensemble de variables. C'est pourquoi des CSP sont souvent utilisés pour modéliser les problèmes étudiés dans plusieurs domaines tels que l'intelligence artificielle et la recherche opérationnelle. Ce mémoire traite de CSP à domaines finis, c'est-à-dire que les domaines des variables sont finis.

Définition 1.1 : Problème de satisfaction de contraintes CSP.

Un CSP est défini par le triplet (X, D, C). Où :

- $X = \{x_1, x_2, .., x_n\}$ est un ensemble de variables.
- $D = \{D_1, D_2, .., D_n\}$ *tel que* $x_i \in D_i$ est l'ensemble des domaines des variables.
- $C = \{c_1, c_2, .., c_n\}$ *tel que* $c_i \subset \prod_{j=1}^{n} D_j$ est l'ensemble de contraintes qui lient les variables.

Définissons d'abord les termes utilisés dans le cadre de l'étude des CSP. Le produit cartésien des domaines des variables est appelé *espace de recherche*; cette mesure est souvent utilisée pour définir la taille du problème étudié. On appelle *instanciation partielle* l'application qui réduit les domaines d'un sous-ensemble de variable X à une seule valeur. Une *instanciation complète* est une application qui réduit les domaines de toutes les variables X à une seule valeur. Une instanciation, partielle ou complète, est dite *cohérente* si toutes les contraintes C sont respectées. Une *affectation partielle* est une instanciation partielle cohérente. Alors qu'une *affectation complète* est une instanciation complète cohérente. Une affectation complète est une solution du CSP. On appelle *sous-problème*, ou *sous-arbre*, l'espace de recherche résultant de l'instanciation d'une ou de plusieurs variables.

On distingue deux types de problèmes étudiés dans le cadre des CSP : les problèmes de satisfaction et les problèmes d'optimisation (CSOP : *Constraint Satisfaction and Optimisation Problem*). Pour les problèmes de satisfaction, on vise l'identification soit de la première solution soit de toutes les solutions. Dans le cas

d'optimisation (CSOP), on cherche la meilleure solution selon certains critères d'optimisation (les critères d'optimisation sont contenus dans une fonction objectif).

Dans ce mémoire, nous traitons des problèmes de satisfaction. Plus précisément, nous cherchons à identifier la première solution dans un contexte où les contraintes exprimées garantissent déjà la qualité de la solution trouvée.

La résolution d'un CSP consiste en la recherche d'une affectation complète. Souvent, l'espace de recherche d'un CSP est structuré sous forme d'un arbre nommé *arbre de recherche*. La résolution du CSP devient alors un problème d'exploration d'un arbre. On distingue deux catégories de méthodes de résolution d'un CSP. Les méthodes incomplètes et les méthodes complètes, dites aussi systématiques. Les méthodes complètes permettent de trouver une solution s'il en existe une, sinon de prouver qu'il n'en existe aucune.

Les méthodes complètes effectuent une exploration complète, implicite, de l'arbre de recherche. C'est-à-dire que la recherche considère tous les nœuds, mais elle ne visite qu'un sous-ensemble des noeuds. Ces méthodes partent d'une affectation vide, où aucune variable n'est instanciée, puis elles construisent pas à pas une affectation complète. L'exploration implicite est obtenue lors de l'élimination des sous-arbres, sous-ensemble de solutions, incohérente vis-à-vis de l'instanciation partielle courante.

Les méthodes incomplètes partent d'une instanciation complète quelconque. Puis elles effectuent des mouvements, des changements d'un sous-ensemble d'instanciations, jusqu'à l'obtention d'une affectation complète. Les méthodes de cette classe sont heuristiques et elles restreignent la recherche à un certain voisinage. Parmi les méthodes de cette classe, on trouve les heuristiques de recherche locale.

La représentation des problèmes sous forme de CSP facilite grandement leur traitement en programmation par contraintes. La section suivante présente le paradigme de programmation par contraintes.

1.2 Programmation par contraintes

Dans cette section nous introduisons la programmation par contraintes (PPC) à travers des généralités et quelques caractéristiques importantes. Puis, nous présentons les éléments clés de ce type de programmation.

Le paradigme de programmation par contraintes s'intègre parfaitement à plusieurs types de programmation, telle que la programmation logique, fonctionnelle, impérative et orientée-objet. Plusieurs logiciels et bibliothèques sont conçus pour intégrer la PPC aux différents types de programmation cités ci-dessus. Par exemple, pour la programmation logique on trouve les logiciels CHIP [46], Eclipse [47], PrologV et Gnu-Prolog [48], alors qu'en programmation fonctionnelle on a le logiciel Oz [49], ensuite dans la

programmation impérative on trouve le logiciel CHOCO [50], puis pour la programmation orientée-objet on a les bibliothèques (C++ et Java) de Ilog Solver [51].

La PPC permet une programmation incrémentale, c'est-à-dire qu'on peut augmenter une application en PPC en y ajoutant des contraintes et des variables. Une autre caractéristique fort intéressante de la PPC est la séparation entre la présentation du problème et sa résolution. En effet, en PPC, on décrit les solutions recherchées et non l'algorithme de résolution à appliquer. Conséquemment, la PPC offre un cadre très souple pour représenter les problèmes combinatoires du monde réel.

La déclaration des solutions, dans la PPC, s'effectue en spécifiant les propriétés des variables et les relations entre variables, exprimées à l'aide des contraintes, que les solutions doivent vérifier. Souvent, les contraintes unaires expriment des propriétés. Par exemple, la contrainte $(x_3 < 4)$ définit 4 comme borne supérieure pour la variable x_3. Alors que les contraintes n-aires établissent des relations entre les variables. De ce fait, les variables elles aussi établissent un lien entre les différentes contraintes. D'ailleurs, ces liens sont à la base du phénomène de propagation des contraintes considéré comme un élément clé en PPC.

La propagation des contraintes exploite les relations logiques entre les variables, dictées par les contraintes, afin de réduire l'espace de recherche. Chaque type de contrainte dispose de son propre algorithme de propagation, dit aussi de filtrage, qui permet l'épuration des domaines des variables liées par ladite contrainte. L'épuration consiste en l'élimination, du domaine de chaque variable, des valeurs qui violent une contrainte compte tenu de l'instanciation partielle courante.

Les algorithmes de propagation utilisent différentes méthodes de filtrage des domaines des variables. Ces algorithmes sont déclenchés lors de divers événements qui surviennent dans les domaines des variables. Le premier est nommé l'événement *'domain'*, surgit lors de tout changement dans le domaine de la variable, par exemple l'élimination d'une valeur. Le deuxième est l'événement *'range'*, surgit lors de changement d'une valeur extrême du domaine de la variable. Le dernier est l'événement *'value'*, surgit lors de l'instanciation d'une variable, réduction du domaine de la variable à une seule valeur.

La communication entre les algorithmes de propagation est assurée à travers les domaines des variables. Ce qui permet à des contraintes de différents types de collaborer. Les algorithmes de filtrage exploitent des mécanismes de cohérence afin de maintenir un certain niveau de cohérence de l'espace de recherche. Les paragraphes suivants présentent quelques mécanismes de cohérence, les algorithmes de propagation courants et l'opération nommée *branchement.*

On distingue généralement les propriétés de cohérence suivantes : la cohérence de nœud, d'arc, de domaine et de bornes. Seules les contraintes unaires peuvent être cohérentes de nœud. Une contrainte unaire est dite cohérente de nœud si la contrainte est vérifiée, autrement dit si toutes les valeurs dans le domaine de la variable respectent la contrainte. Cette propriété s'obtient en éliminant du domaine de la variable les valeurs qui violent la contrainte unaire concernée.

Pour les contraintes binaires, on parle de cohérence d'arc. Étant donnée une contrainte binaire $c(x, y)$ liant les variables x *et* y dont les domaines sont respectivement D_x *et* D_y. La cohérence d'arc est atteinte si :

- $\forall v \in D_x$, il existe (v,v') ∈ c où v' ∈ D_y et
- $\forall$ v ∈ D_y, il existe (v',v) ∈ c où v ∈ D_x.

Autrement dit, les domaines de x *et* y ne contiennent que des valeurs qui participent à au moins une solution vis-à-vis de l'instanciation partielle courante. La cohérence d'arc permet d'effectuer plus de filtrage que la cohérence de noeud, en nécessitant un faible coût supplémentaire en calcul.

La cohérence de domaine est une cohérence d'arc généralisée, appliquée aux contraintes n-aires avec n >2. Mais cette propriété est souvent coûteuse et difficile à atteindre. Alors que la cohérence de bornes [37] concerne les valeurs minimums et maximums des domaines, comme son nom l'indique. Ce type de cohérence s'applique aux contraintes >, <, !, et = qui comparent des expressions algébriques liant plusieurs variables. Une contrainte est dite cohérente de bornes si toutes les bornes des variables participantes ont un support. Autrement dit, ils font partie d'une solution vis-à-vis l'instanciation partielle courante.

L'algorithme de propagation le plus simple est le *backtracking*. Il applique la cohérence de domaine sur les contraintes dont toutes les variables sont instanciées. C'est un algorithme peu coûteux, mais aussi peu performant puisqu'il ne détecte les incohérences que lorsqu'elles apparaissent. Le *forward checking* est un algorithme de

propagation qui permet de prévoir les incohérences futures. Il applique la cohérence de domaine aux contraintes qui portent sur la variable courante, la variable traitée, et au plus une variable future. L'algorithme *looking ahead* ou *Maintaining Arc Consistency* (MAC) filtre davantage que l'algorithme précédent parce qu'il applique la cohérence de domaine sur les contraintes qui lient toutes les variables futures. Toutefois, le MAC est plus coûteux, en temps CPU, puisqu'il demande plus de travail à chaque noeud de l'arbre de recherche. La figure 1.1 montre les différences entre le *backtracking*, le *forward checking* et le *Maintaining Arc Consistency*.

La recherche de solution en PPC consiste en une exploration implicite de l'arbre de recherche. Cette exploration est basée sur l'opération nommée branchement (figure 1.2). Le branchement permet la division de l'espace de recherche en sous-espaces. Souvent les sous-espaces générés ne sont pas tous de la même taille. Cependant, ils forment une partition de l'espace courant, sinon la recherche est redondante ou incomplète. Principalement, deux choses définissent le branchement : le "où" (le lieu de branchement) et le "quoi" (la partie à traiter). Bref, le branchement détermine quel sous-arbre doit être traité.

La stratégie de recherche est l'ensemble des procédures de choix utilisées lors du branchement. La stratégie de recherche est un élément central dans une application en PPC. La section suivante décrit cet élément de la programmation par contraintes.

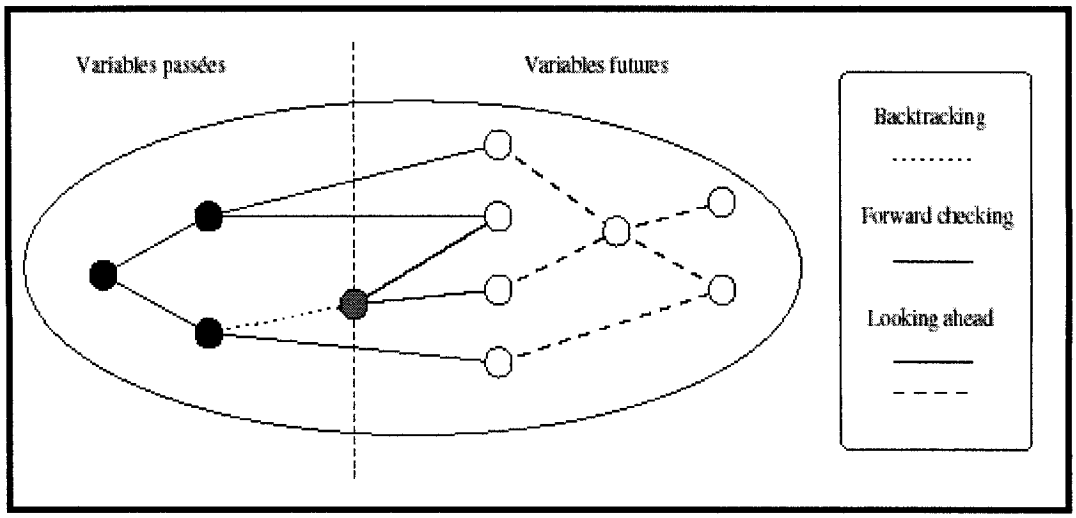

Figure 1.1 : Cohérence d'arc [1]

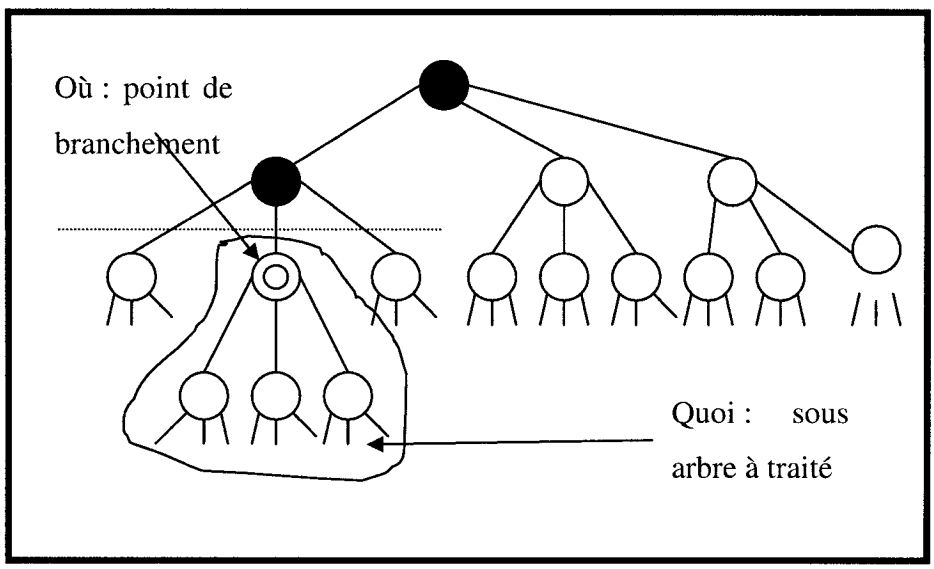

Figure 1.2 : opération branchement

1.3 Stratégie de recherche

La résolution des problèmes NP-Difficile constitue un défi pour tous les paradigmes de programmation, la PPC n'y échappe pas. Dans cette catégorie de problèmes, l'espace de recherche est énorme. Par exemple, pour certains jeux utilisés dans ce mémoire, cet espace atteint 10^{950}. Donc, une stratégie de recherche naïve ou peu informée n'est souvent pas performante, surtout lorsque le problème admet très peu de solutions. D'où l'importance accordée à la conception d'une stratégie de recherche efficace.

Brièvement, une stratégie de recherche consiste en trois éléments. Le premier est le choix du type de parcours de l'arbre de recherche. Le second est le choix de la variable à instancier. Le dernier est le choix de la valeur à affecter. La conception d'un algorithme exact, qui trouve la solution optimale tout le temps, pour la stratégie de recherche est souvent compliquée. D'où le recours aux heuristiques qui sont tirées de l'expérience et d'analogies plutôt que d'une analyse scientifique trop complexe qui doit évaluer le maximum d'éléments.

Concernant le type de parcours de l'arbre de recherche, on utilise souvent une recherche en profondeur d'abord (DFS : Depth First Search). Cette méthode consiste en une exploration des nœuds gauches en premier, les nœuds de gauche représentant les choix effectués par l'heuristique, autrement dit, dans ce type de parcours on fait totalement confiance à l'heuristique.

D'autres méthodes de parcours existent, mais généralement elles ne constituent que des variantes plus ou moins sophistiquées de la méthode DFS. On peut citer par exemple la recherche avec divergence limitée LDS (Limited Discrepancy Search), qui est une méthode DFS permettant un nombre limitée de divergences, une divergence correspond au choix d'un nœud autre que celui de gauche, donc quelques dérogations vis-à-vis les recommandations de l'heuristique utilisée. Un inconvénient majeur de cette méthode est la redondance dans la génération des ordres. Une autre méthode nommée DDS (Depth-bounded Discrepancy Search) est une variante de la méthode LDS. Cette méthode limite la profondeur, de l'arbre, à laquelle les divergences sont autorisées.

Le deuxième élément de la stratégie de recherche est l'ordre d'instanciation des variables. Cet ordre influence la topologie de l'arbre de recherche et donc sa taille. Généralement, on vise la construction de l'arbre de recherche le plus petit possible pour accélérer la recherche. Donc, pratiquement toutes les heuristiques de choix de variable sont basées sur le principe d'échec d'abord FF (***Fail Fir***st). Dans ce type d'heuristiques, on explore les variables les plus susceptibles d'être impliquées dans un échec en premier, afin d'effectuer le meilleur élagage possible et réduire rapidement la taille de l'arbre de recherche. L'élagage est la suppression des branches incohérentes avec l'instanciation partielle courante.

Le troisième élément de la stratégie de recherche est l'ordre de choix de la valeur. Celui-ci influence l'ordre dans lequel les branches de l'arbre de recherche seront explorées. Donc, l'ordre et la vitesse auxquels les solutions seront trouvées. Subséquemment, celui-ci devient décisif lorsqu'on cherche une seule solution (c'est le cas qui nous intéresse dans ce mémoire), la meilleure solution ou l'amélioration de la vitesse de résolution.

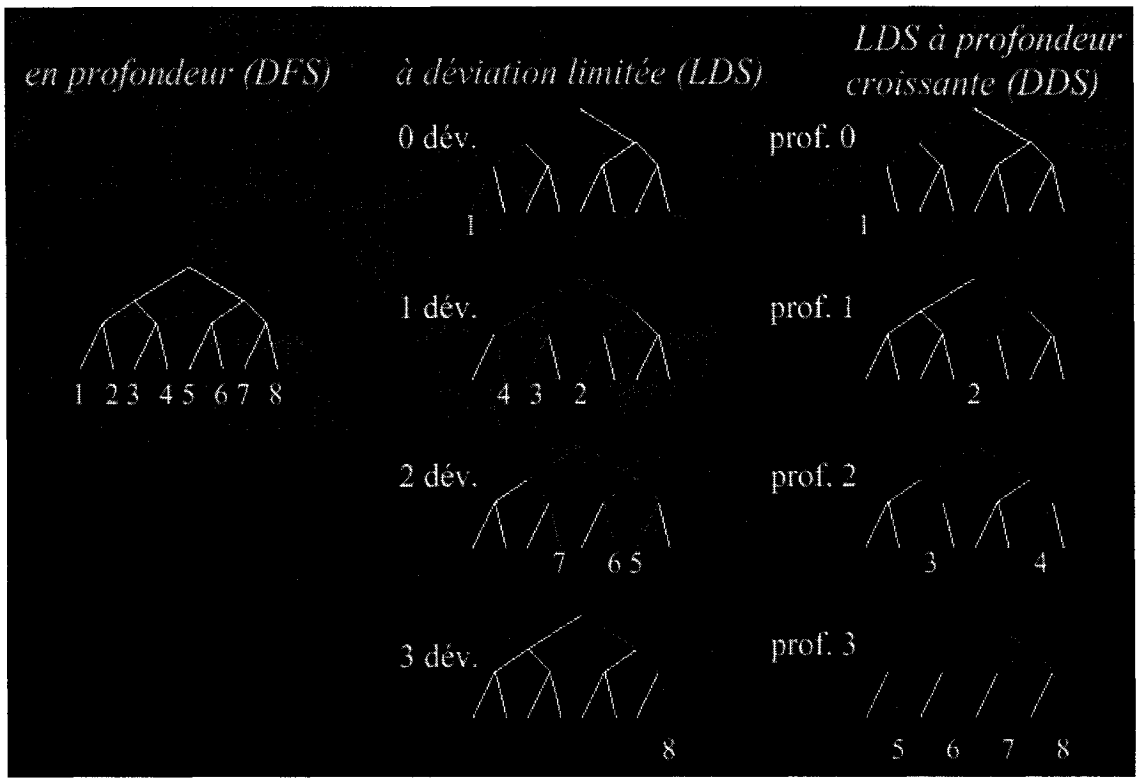

Figure 1.3 : exemples de parcours DFS, DDS et LDS. [31]

À l'inverse des heuristiques de choix de variable, les heuristiques de choix de valeur sont basées sur le principe du succès d'abord SF (*Succeed First)*. Ce principe vise l'exploration des branches les plus prometteuses, soit celles qui conduisent rapidement aux bonnes solutions. L'heuristique de choix de valeur idéale est celle qui permettra de trouver une solution sans aucune remise en question des choix antérieurs (retour arrière : *backtracking)*. Néanmoins, une telle heuristique est souvent difficile à concevoir.

On peut diviser les heuristiques de choix de valeur en deux catégories : statiques et dynamiques. Les heuristiques statiques établissent un ordre avant le début de la recherche. Cet ordre est souvent basé sur la structure particulière du problème considéré et l'état initial de la résolution. Les heuristiques statiques sont souvent très peu performantes puisqu'elles ne prennent pas en considération les changements qui s'opèrent lors de la résolution. On entend par changements, principalement, les changements des domaines des variables.

Les heuristiques dynamiques offrent un choix plus intéressant. Ces heuristiques établissent l'ordre pendant la recherche. Elles utilisent des informations concernant l'évolution de la résolution, telles que les domaines des variables, les états des contraintes, la fonction objective, etc. Ces informations permettent d'effectuer des approximations et des prévisions afin d'anticiper les états du système.

Généralement, les heuristiques dynamiques réussissent mieux que les heuristiques statiques, mais elles sont plus difficiles à concevoir et souvent plus consommatrices en temps CPU. Cependant, Frost [53] montrent que ce surcoût s'avère bénéfique lorsque le problème traité est suffisamment difficile. Dans ce qui suit, nous énumérons les heuristiques de choix de valeur statiques et dynamiques existantes dans la littérature au mieux de notre connaissance.

1.4 Revue de la littérature

1.4.1 Heuristiques de choix de valeur statiques

Dans la littérature, on trouve des heuristiques statiques qui implémentent un ordre aléatoire, lexicographique, ascendant (plus petite valeur d'abord), descendant (plus grande valeur d'abord) ou un ordre fixe basé sur la structure particulière du problème étudié. Les quatre ordres (aléatoire, lexicographique, ascendant, descendant) cités ci-dessus sont souvent intégrés dans les outils de programmation par contraintes. Ces derniers représentent des bons outils pédagogiques et utilitaires à la conception de stratégies plus élaborées. Utilisées toutes seules, comme heuristiques de choix de valeur, elles donnent des performances limitées lors de la résolution des problèmes réels.

Les heuristiques statiques qui exploitent des structures particulières sont souvent plus efficaces, mais nécessitent une connaissance approfondie du problème étudié. Elles exploitent la structure particulière du problème étudié.

D'autres heuristiques exploitent des concepts généraux tels que les conflits, les supports, etc. Par exemple, l'heuristique SLC (Static least-conflicts) Frost [52] est basée sur le nombre de conflits auxquels chaque valeur participe. Deux couples (variable, valeur) distincts C1 et C2 sont en conflit s'ils ne peuvent pas faire partie d'une même solution selon l'affectation partielle. Le nombre de conflits auxquels une valeur participe

est le nombre de couples (variable, valeur) que le choix de cette valeur rendra incompatibles. Les calculs sont effectués sur les domaines initiaux (avant le début de la recherche) et l'ordre obtenu est utilisé pendant toute la recherche. L'heuristique SLC choisit les valeurs les plus prometteuses, soit celles ayant le nombre de conflits le plus petit.

L'heuristique SMP (Static max-promises) exploite le concept de support. Le support d'un couple (variable, valeur) (V1, v1) vis-à-vis une variable V2 est l'ensemble des valeurs, contenues dans le domaine de V2, cohérentes avec l'affectation V1 = v1. La SMP calcule pour chaque couple (variable, valeur) le produit des supports. Le résultat est une estimation statique du nombre de solutions auxquelles le couple participe. Là aussi, les calculs sont effectués sur les domaines initiaux (avant le début de la recherche) et l'ordre obtenu est utilisé pendant toute la recherche. L'heuristique choisit alors la valeur qui a le produit le plus grand, soit celle qui participe au plus grand nombre de solutions.

1.4.2 Heuristiques de choix de valeur dynamiques

Généralement, les heuristiques statiques restent peu performantes dans la résolution de problèmes réels complexes. Les heuristiques dynamiques offrent alors une alternative plus prometteuse. Ces heuristiques utilisent différentes informations sur l'évolution de la résolution pour effectuer des calculs, des approximations et des prévisions.

Les domaines des variables, qu'elles soient principales ou non, constituent la source d'informations la plus utilisée dans la conception des heuristiques de choix de valeur dynamiques. Notons qu'une variable principale est une variable qu'on cherche à instancier (à résoudre). Frost et Dechter [43] proposent différentes heuristiques basées sur le concept de conflit. La principale est l'heuristique LVO (*look-ahead value ordering*), elle utilise le mécanisme *forward checking*. La LVO est une SLC dynamique, elle instancie chaque valeur de la variable courante et examine l'effet de la propagation et du filtrage sur les domaines des autres variables. L'heuristique choisit la valeur qui réduit le moins l'espace de recherche, soit celle qui a le plus de chance d'aboutir à une solution.

Frost et Dechter [43] utilisent, dans leur benchmark interne, l'heuristique LVO-MC qui sélectionne la valeur ayant la plus grande somme des supports. Larrosa et Meseguer [44] proposent une heuristique proche de la LVO-MC, basée sur le concept du support, nommée *highest support*.

L'apprentissage est un concept qui est exploité par quelques auteurs. Dans ce cas, souvent on enregistre des informations, lors d'instanciations ou d'exécutions antérieures, qui seront utilisées par la suite pour effectuer les choix de valeur. Ci-dessous, nous citons quelques heuristiques utilisant le concept d'apprentissage.

Frost et Dechter [35] proposent l'heuristique ***Sticking value*** (valeurs collantes). Lorsqu'un retour arrière est effectué, on enregistre les anciennes valeurs choisies. Ces valeurs seront testées en premier lors des prochains choix. On estime que les valeurs qui étaient bonnes auparavant le seront encore.

Une autre heuristique, basée spécifiquement sur le principe d'apprentissage par l'échec, était proposée par Prcovic et Neveu [4]. La phase d'apprentissage est obtenue par une recherche alternée dans les sous-arbres, ce qui permet de mettre à jour la mesure ABL (longueur moyenne des branches). L'ABL est la longueur entre le sommet du sous-arbre et un échec, cette mesure est calculée pour chaque valeur dans le domaine de la variable courante. Rappelons qu'un sous-arbre (ou sous-problème) consiste en l'arbre de recherche résultant lors du choix d'une valeur de la variable courante. L'heuristique choisit la valeur ayant la plus grande moyenne des ABL.

Refalo [5] propose une heuristique basée sur l'apprentissage et les domaines des variables non encore instanciées. Cette heuristique utilise le concept d'impact introduit par l'auteur, mais s'inspirant de la programmation mathématique. L'impact est une mesure de réduction de la taille de l'espace de recherche. La réduction est calculée à partir des tailles des domaines des variables avant et après pour chaque affectation. L'auteur a remarqué que la distribution de l'impact d'un certain choix (couple variable valeur) ne varie pas trop d'un nœud à un autre. En plus, cette variation présente une distribution uniforme avec un pic dans la valeur moyenne. Donc, à chaque nœud au lieu de recalculer l'impact au complet, ce qui introduirait des temps de calcul très grands, on n'effectue que le calcul de la moyenne des anciennes valeurs de l'impact. Puisqu'il s'agit d'une méthode d'apprentissage, la recherche est réinitialisée selon un certain schéma afin de profiter des impacts des affectations accumulées et faire de meilleurs choix en haut de l'arbre de recherche. Refalo [5] a utilisé l'impact aussi pour le choix de la variable. Il propose deux façons de calculer l'impact d'une variable soit la somme des impacts de ses valeurs ou leurs produits. Il conclut que l'utilisation de la somme donne de meilleurs résultats.

Comme nous l'avons indiqué auparavant, les heuristiques de choix de valeur visent le succès d'abord (SF : *Succeed First*). Les valeurs participantes à un grand nombre de solutions sont donc de bonnes candidates. En effet, quelques auteurs ont proposé des heuristiques basées sur le nombre de solutions potentielles. Dechter et Pearl [28] ainsi que Sadeh et Fox [45] proposent trois versions d'heuristiques de choix de valeur qui utilisent des techniques de relaxation des sous-problèmes restants afin d'effectuer l'estimation du nombre des solutions potentielles.

Dans le même sens, Elbassioni et Katriel [6] ont introduit la mesure k-multiconsistence autour de laquelle ils ont proposé une heuristique générique. Un couple variable valeur (Vi,j) est dit k-multiconsistent s'il existe K solutions distinctes dans lesquelles l'affectation (Vi = j) est cohérente. Donc, k est une mesure de la robustesse de l'affectation (Vi = j). L'heuristique choisirait les affectations atteignant la k-multiconsistence pour un k donné. Les auteurs proposent quelques algorithmes qui permettent la détermination de la k-multiconsistence pour les variables participantes aux contraintes globales (***AllDifferent'***, (***Cardinality)*** et (***Same).*** Nous pensons que la mesure k-multiconsistence constitue un critère heuristique fort intéressant pour le cas qui nous intéresse ici, soit le choix de valeur dans le cadre d'une application en PPC.

Récemment, Pesant [3] a proposé une nouvelle technique qui fournit une meilleure approximation du nombre de solutions. L'estimation calculée pourra être utilisée facilement, entre autres, dans des heuristiques de choix de valeur robustes et modulaires. Les algorithmes exacts proposés sont basés sur la structure particulière de chaque contrainte prise séparément. L'auteur donne des algorithmes polynomiaux et des pistes pour calculer cette estimation pour les variables participantes aux contraintes binaires, binaires arithmétiques, linéaires, ainsi que quelques contraintes globales.

Souvent, les heuristiques de choix de valeur, statiques ou dynamiques, s'inspirent de la structure du problème étudié. Donc, elles sont spécialisées et taillées pour un problème donné. En conséquence, elles sont difficilement réutilisables. C'est pourquoi ce travail de recherche vise la conception et l'intégration d'heuristique robuste, modulaire et générique.

Les travaux de Pesant [3] et Elbassioni et Katriel [6] partagent une caractéristique très importante avec ce que nous proposons dans ce mémoire. Elles sont centrées sur les contraintes, ce qui les rend génériques et modulaires. En effet, un point commun de tous les problèmes traités en programmation par contraintes est le type de contraintes utilisées. Donc, les algorithmes exploitant les structures particulières des contraintes séparément fourniront des heuristiques de choix de valeur modulaires et génériques.

Chapitre 2: Contexte d'application

Dans ce chapitre nous présentons l'application de validation de ce travail de recherche. Nous commençons par présenter le problème de validation choisi, soit la génération des horaires. Puis, nous exhibons l'application de validation : Hibiscus. Cette application permet la génération d'horaire pour le personnel d'un hôpital.

2.1 Les problèmes de génération d'horaire

L'augmentation de la demande, autant quantitative que qualitative, dans le secteur canadien de la santé ainsi que la rareté du personnel qualifié ont mis ce secteur en déficit accru et prolongé. Cette crise exerce une grande pression sur les ressources humaines, technologiques et financières du secteur. L'enjeu devient alors la bonne gestion des ressources pour atteindre les objectifs en ce qui concerne la qualité des soins, les coûts ainsi que la satisfaction du personnel. D'où le grand intérêt que le problème de confection d'horaire du personnel médical génère [1, 2, 7, 8, 9, 10] et le nombre grandissant d'applications commerciales dédiées à ce sujet [36, 38, 39, 40, 41, 42].

Les caractéristiques particulières du secteur de la santé rendent le problème de confection d'horaire difficile. Tout d'abord, le service doit être assuré 24 heures par jour. Puis la demande varie énormément, sans préavis, de jour en jour et même d'heure en

heure. En plus, il faut répondre à des besoins variés, sans oublier la pénurie du personnel médical qualifié.

La génération d'un bon horaire doit respecter, entre autres, les droits et conventions de travail, les préférences du personnel ainsi que les règles ergonomiques et d'équilibrage. Un bon horaire permet l'augmentation du niveau d'équité, donc la réduction des sentiments d'injustice ainsi que des manifestations d'insatisfaction telles que les grèves, l'absentéisme, la baisse du rendement, etc. Subséquemment, l'exploitation des ressources humaines, technologiques et financières est améliorée, ce qui permet l'amélioration de la qualité des services offerts.

Plusieurs approches sont utilisées dans la résolution des problèmes de confection d'horaire. Par exemple, dans la programmation mathématique on trouve, entre autres, Forget [15], Beaulieu [8], Jaumard et coll. [17], Monaci et Toth [10]. En métaheuristiques on trouve, entre autres, Thompson [22] et Buzòn [9]. En programmation par contraintes on a, entre autres, Bourdais [1], Heus [16], Abdennadher et Schenker [7], Darmoni et coll. [13], Caseau et coll. [11], Meisels et coll. [18], Cheng et Lee [12], Saadie [21], Feuilletin [14]. On trouve aussi des hybridations de la programmation par contraintes avec diverses heuristiques et métaheuristiques. Citons par exemple, Rousseau, Pesant et Gendreau [20], Meyer [19], Burke et coll. [23] et [24], Ross et coll. [25].

2.2 Description du problème de confection d'horaire

La confection d'horaires est un problème d'affectation de ressources à un ensemble de tâches en respectant des contraintes (dites règles). La conception d'horaire consiste en l'affectation des ressources aux différentes tâches en satisfaisant les contraintes. Souvent, on s'intéresse aussi à la génération d'un bon horaire vis-à-vis certains critères.

Trois types d'horaire existent : les horaires cycliques avec rotations, les horaires cycliques sans rotation et les horaires non cycliques. Les horaires cycliques avec rotation sont construits à partir de patrons d'horaires sur une période courte, que les membres du personnel suivent à tour de rôle. Ce type d'horaire garantit une équité parfaite, mais il ne permet aucune flexibilité. Les horaires cycliques sans rotation sont également construits à partir de patrons d'horaires personnalisés, chaque membre du personnel possède son propre patron, sur une courte période. Ce type d'horaire conserve l'équité et permet une certaine flexibilité, mais ne permet pas de prendre en compte les préférences épisodiques (périodes de vacances, indisponibilités occasionnelles,...). Finalement, les horaires non cycliques sont reconstruits au début de chaque période de planification (nommée horizon de planification). Ce type d'horaire permet une flexibilité maximale, mais un haut niveau d'équité est difficile à atteindre. Les méthodes de génération d'horaires non cycliques sont les plus difficiles à concevoir vu le nombre important de contraintes. Ce mémoire traite de la conception d'horaires non cycliques.

En gros, quatre éléments permettent de définir un problème de confection d'horaire, soit les ressources, les tâches, l'horizon et les règles. Ci-dessous, une brève description de chacun de ces éléments :

- Les ressources : Il s'agit du personnel médical. Dans notre cas, ce sont les infirmières et les infirmiers.

- Les tâches : il s'agit d'une période de travail indivisible pouvant être accomplie par une seule personne au cours d'une journée. Elle est identifiée par un symbole, une heure de début, une heure de fin et un nombre d'heures payées. L'ensemble des tâches comprend les quarts de travail, les séances de cours et les réunions. Les deux dernières tâches sont très rares.

- Les règles : ce sont les contraintes prises en compte complètement ou partiellement (contrainte dure ou souple). Dans notre cas, les règles modélisées ne peuvent jamais être prises en compte toutes simultanément parce que certaines d'entre elles sont contradictoires ou redondantes.

- L'horizon de planification : est la période du calendrier qui est concernée par l'horaire, autrement dit la période du calendrier couverte par l'horaire. Il est identifié par une date de début et une date de fin.

2.3 Hibiscus

2.3.1 Généralités

Dans ce qui suit, nous présentons l'application Hibiscus [2]. C'est une application de planification d'horaire des infirmières et des médecins. Elle se veut modulaire, facilement adaptable aux différents contextes rencontrés dans le secteur de la santé. Hibiscus est basé sur la programmation par contraintes (ILogSolver 4.4). La conception de ce système a débuté avec les travaux de William Feuilletin [54] en 2000, suivi de ceux de Julien Félisiak et coll [55], et de Stéphane Bourdais [1].

Hibiscus génère des horaires non cycliques. C'est la façon la plus flexible, mais c'est aussi la méthode la plus exigeante, vu le nombre important de contraintes qu'il faut respecter. Hibiscus permet la prise en compte de cinq catégories de contraintes : demande (DEM), charge de travail (WOR), disponibilité (AVA), ergonomie (ERG) et équité (FAI).

La catégorie demande (DEM) concerne les contraintes assurant la couverture minimale des besoins par période par jour et par niveau de personnel. La charge de travail (WOR) touche les contraintes définissant le nombre d'heures, ou de quarts, que chaque personne, infirmière ou médecin, doit travailler. Dans la catégorie (AVA) on trouve les règles définissant les disponibilités de chaque membre du personnel, tel que les quarts réalisables, les préaffectations et les affectations interdites. La catégorie ergonomique (ERG) contient les règles de gestion des fins de semaines, des vacances, de succession

des quarts, etc. La classe équité (FAI) vise l'équilibrage de certaines caractéristiques des horaires du personnel.

Hibiscus intègre plusieurs stratégies de recherche telles que les stratégies de base (aléatoire, lexicographique, plus petit domaine d'abord, plus grand domaine d'abord...) ainsi que d'autres heuristiques spécialisées. Par exemple, l'attribution des fins de semaines d'abord, l'ordonnancement des infirmières selon le nombre de solutions trouvées pour chacune pendant un laps de temps fixe...

2.3.2 Modèle de PPC

Hibiscus utilise un modèle classique pour la représentation du problème de génération des horaires. La solution est représentée sous forme d'une grille en deux dimensions. La première dimension est le personnel alors que la deuxième est l'horizon de planification, voir figure 2.1.

Chaque case correspond à une variable de décision $X_{i,j}$ qui définit le quart travaillé (jour, soir...) par la personne i le jour j. Les variables sont liées verticalement et horizontalement par les contraintes. Une contrainte verticale lie un sous-ensemble du personnel à un jour donné, alors qu'une contrainte horizontale concerne une seule personne et un sous-ensemble de jours.

Les seules contraintes verticales sont les contraintes de satisfaction de la demande. Toutes les contraintes restantes, dites horizontales, sont les contraintes relatives à la qualité des horaires individuels. Parmi ces dernières, on trouve par exemple les contraintes de respect de la charge de travail et de l'équilibrage des types des quarts. Toutes les contraintes, qu'elles soient horizontales ou verticales, sont en concurrence les unes avec les autres.

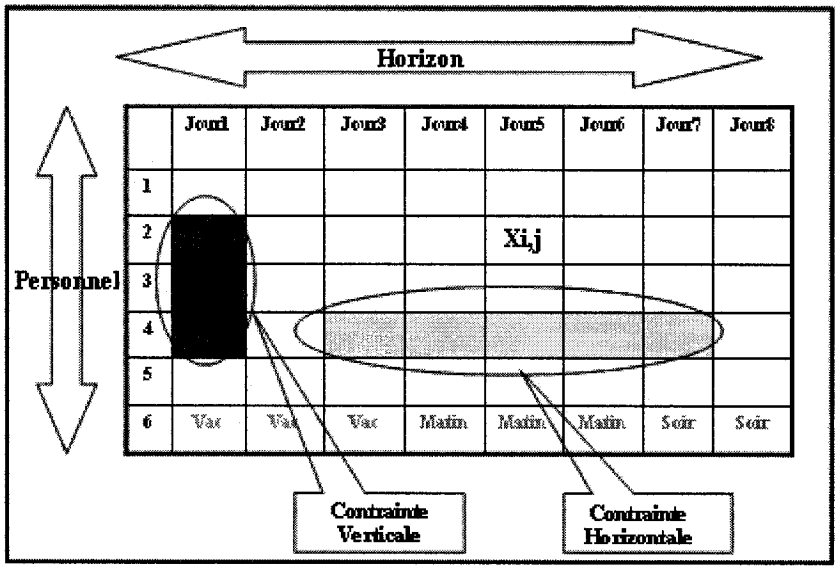

Figure 2.1 : Modèle utilisé pas Hibiscus.

On peut classer l'ensemble des contextes médicaux rencontrés sous deux familles bien distinctes. Ceux dont les contraintes verticales sont serrées (marge de manœuvre restreinte) alors que les contraintes horizontales de charge de travail sont lâches, marge de manœuvre importante. Puis ceux dont les contraintes verticales sont lâches et dont les contraintes horizontales de respect de la charge de travail sont serrées.

La première famille de problèmes correspond aux contextes des salles d'urgence. Tandis que la seconde correspond plutôt aux unités de soins infirmiers. Hibiscus propose, pour les deux contextes, deux décompositions : décomposition par horaire individuel et décomposition par jour, voir figure 2.2.

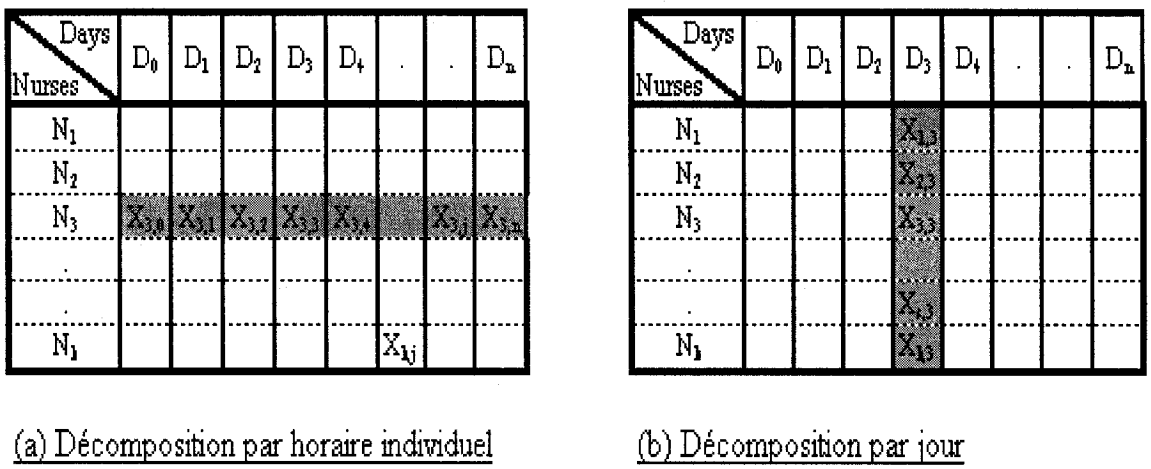

Figure 2.2 : Décompositions du problème.

La décomposition par horaire (figure 2.2 a) remplit la grille ligne par ligne, soit infirmière par infirmière (en anglais : NBN pour *nurse by nurse*), donc les contraintes horizontales sont traitées en priorité. Alors que la décomposition jour par jour (figure 2.2 b) (en anglais : DBD pour *day by day*), comme son nom l'indique, procède au remplissage de la grille jour par jour. Dans ce cas, ce sont les contraintes verticales qui sont traitées en priorité.

Dans les unités de soins infirmiers, les contraintes horizontales sont plus nombreuses et plus serrées, surtout celles du respect de la charge de travail. Donc, on adopte la décomposition par horaire individuel parce qu'elle permet de traiter les

contraintes horizontales en priorité. Dorénavant, la désignation décomposition infirmière par infirmière (NBN) fera référence à la décomposition par horaire individuel.

Chapitre 3 : Heuristique d'incitation

Dans ce chapitre, nous présentons le principe général de l'heuristique d'incitation proposée, puis la contrainte Cardinalité ainsi que l'application de l'heuristique d'incitation à la contrainte Demande. Ensuite nous abordons la contrainte de Cardinalité pondérée ainsi que l'application de l'heuristique d'incitation à celle-ci.

3.1 Principe général

Nous partons du fait que la résolution d'un CSP doit respecter impérativement toutes les contraintes du problème. Donc, nous allons effectuer le suivi de la satisfaction de chaque contrainte séparément et essayer de maintenir toutes les contraintes dans un même niveau de satisfaction.

À certaines contraintes du modèle, nous associons une heuristique d'incitation dont le but est de favoriser les choix de valeur (affectations) qui sont plus susceptibles de participer à la satisfaction de la contrainte concernée. L'heuristique d'incitation favorise une affectation en lui attribuant un score. La définition (3.1) ci-dessous donne le schéma général d'une heuristique d'incitation.

Nous utilisons les notations suivantes :

- $X = \{x_1, x_2, .. x_m\}$: Variables du problème,
- $V = \{v_1, v_2, ... , v_n\}$: Vecteur des valeurs,
- $T = \{t_1, t_2, ..., t_k\}$ *avec* $t_i \subset V \ \forall \ 1 \leq i \leq k$: Ensemble des types, chaque valeur $v \in V$ appartient à un type t_i, T ne forme pas nécessairement une partition de V,
- $D_{x_i} = \{v_{i_1}, v_{i_2}, .. v_{i_b}\}$ *avec* $D_{x_i} \subset V$: domaine de la variable X_i,
- A : l'ensemble des affectations possibles, autrement dite les couples (variable, valeurs) possibles :

$$A = \bigcup_{i=1}^{m} \{x_i\} \times D_{x_i},$$

$$A = \{a_1, a_2, .. a_r\},$$

avec $a_i = (x_s, v_j)$ *tel que* $x_s \in V$ *et* $v_j \in D_{x_s}$,

a_i est l'affectation de la valeur v_j à la variable x_s.

Définition 3.1 : Heuristique d'incitation.

Une heuristique d'incitation '*h*' est définie par un couple (C_h, w_h) tel que C_h est la contrainte à laquelle *h* est liée et w_h est la pondération de '*h*'. L'heuristique '*h*' dépend d'un sous-ensemble de variables du problème $X_h \subset X$. '*h*' favorise (on attribuant un score positif) un sous-ensemble des affectations possibles $A_h \subset A$ selon l'état de la recherche $(A_h = \bigcup_{N} \{x_i\} \times D_{x_i} \ \forall x_i \in V_h)$. On entend par état de la recherche principalement l'état des domaines des variables non encore instanciées ainsi que l'instanciation partielle courante.

Nous définissons ϕ_h la fonction qui associe les incitations aux affectations possibles, tel que :

$$\Phi_h : A_h \xrightarrow{\Phi_h} \mathbb{R} \qquad (3.1)$$

Soit $H = \{h_1, h_2, .., h_n\}$ l'ensemble des heuristiques d'incitation actives. Toute affectation a_i se voit attribuer un score cumulé $\pi_c(a_i)$ par les heuristiques telle que :

$$\pi_c(a_i) = \sum_{h \in H} w_h \phi_h(a_i) \qquad (3.2)$$

Rappelons que w_h est la pondération, nommée aussi poids, de l'heuristique h. Cette pondération permet d'établir une hiérarchie sur l'ensemble des heuristiques H.

L'affectation choisie par la stratégie de recherche est celle dont le score cumulé π_c est le plus élevé. L'équation (3.2) montre la collaboration des différentes heuristiques, à travers leurs incitations $\phi_h(a_i)$ et leurs pondérations w_h, dans l'attribution du score cumulé qui définit le choix de la valeur. La collaboration des heuristiques permet de diriger la recherche vers les zones qui augmentent le niveau de satisfaction globale de toutes les contraintes du problème.

3.2 Contrainte Cardinalité

Dans cette section, nous présentons le premier type de contrainte étudiée dans ce travail de recherche, soit la Cardinalité. Nous avons choisi cette contrainte parce que le problème concerné, la génération d'horaire, en contient un nombre important. Citons par exemple la demande, la charge de travail, les vacances, etc.

3.2.1 Cardinalité

Une contrainte de Cardinalité impose une limite sur le nombre d'occurrences d'un type donné. Dans notre cas elle impose une limite sur le nombre d'occurrences d'une certaine valeur au sein d'un sous-ensemble de variables X. L'expression de cette contrainte peut être obtenue facilement à l'aide d'opérateurs mathématiques d'égalité (=) et d'inégalité $(<, \leq, >, \geq)$. Mais, les contraintes exprimées de la sorte ne permettent pas un très bon filtrage. Souvent, on exprime les contraintes de Cardinalité à l'aide de la contrainte globale GCC (*Global Cardinality Constraint*). Une définition formelle de la contrainte Cardinalité est donnée ci-dessous (Définition 3.2).

Définition 3.2 : Contrainte de Cardinalité.

Une contrainte de Cardinalité C_c est un quadruplet $(Y, t, \bar{v}, \underline{v})$ tel que $Y = \{x_i, x_j, ..., x_n\}$, $Y \subset X$ l'ensemble des variables concernées par la contrainte, t est le type concerné par la contrainte, $\bar{v}$ la borne maximale et $\underline{v}$ la borne minimale de la contrainte. La contrainte Cardinalité impose que le nombre de variables $x_i \in Y$ prenant leurs valeurs dans t soit compris entre $\bar{v}$ et $\underline{v}$.

Exemple 3.2 :

Considérant les données suivantes :

- $X = \{x_1, x_2, x_3, x_4\}$: ensemble des variables.
- $V = \{1,2,3,4,a,b\}$: vecteur des valeurs.
- $D_{x_1} = D_{x_2} = D_{x_3} = D_{x_4} = \{1,2,3,4,a,b\}$: domaine des variables.
- $T = \{t_1, t_2\}$ *avec* $t_1 = \{1,3\}$ *et* $t_2 = \{2,4\}$: ensemble des types.

Une contrainte de Cardinalité $C_{c1}(\{x_1, x_2, x_4\}, t_1, 2, 1)$ impose que parmi les variables x_1, x_2 *et* x_4 le nombre de variables ayant une valeur impaire (t_1) doit être compris entre 1($\underline{v} = 1$) et 2 ($\overline{v} = 2$). Le tableau 3.1 montre l'état initial des domaines des variables.

Tableau 3.1 : Domaines initiaux des variables de l'exemple 3.2

Variable	x_1	x_2	x_3	x_4
Domaine	$\{1,2,3,4,a,b\}$	$\{1,2,3,4,a,b\}$	$\{1,2,3,4,a,b\}$	$\{1,2,3,4,a,b\}$

3.2.2 Heuristique d'incitation pour la contrainte Cardinalité

Dans cette section, nous commençons par donner une définition de l'heuristique d'incitation pour une contrainte de Cardinalité. Puis, nous détaillons les étapes effectuées par cette heuristique.

Définition 3.3 : Heuristique d'incitation pour une contrainte de Cardinalité.

Une heuristique d'incitation Cardinalité h_c est définie par le couple (C_c, w_{h_c}) tel que :

- C_c : La contrainte de Cardinalité à laquelle l'heuristique h_c est liée,
- w_{h_c} : Le poids de l'heuristique h_c.

Nous avons défini trois états pour l'heuristique d'incitation, soit O^-, I et O^+. À chaque événement dans le domaine d'une variable de V_h, l'heuristique effectue les mises à jour suivantes :

1. La mise à jour des paramètres $\bar{n}$, $\underline{n}$ et $\tilde{n}$ tel que :
   a. $\bar{n}$ le nombre d'occurrences participantes gagnées,
   b. $\underline{n}$ le nombre maximal d'occurrences participantes possible,
   c. $\tilde{n}$ la prévision.
2. La mise à jour de son état,
3. La mise à jour des incitations à appliquer.

3.2.2.1 Mise à jour des paramètres

Notons Θ la fonction qui définit le type de contribution de chaque variable de V dans la satisfaction de la contrainte $C_c(Y, t, \bar{v}, \underline{v})$. Θ est définie comme suit :

$$\Theta: x_i \rightarrow \{\{0\},\{1\},\{0,1\}\} \text{ tel que: } \begin{cases} \Theta(x_i)=\{0\} & si\ D_{X_i} \cap t=\{\} \quad cas\ 1. \\ \Theta(x_i)=\{1\} & si\ D_{X_i} \subseteq t \quad cas\ 2. \quad \forall x_j \in Y \\ \Theta(x_i)=\{0,1\} & si\ non \quad cas\ 3. \end{cases} \qquad (3.3)$$

Dans l'équation (3.3), le premier cas (cas 1) concerne les variables dont les domaines ne contiennent aucune occurrence du type concernée par la contrainte C_c. Donc, ces variables ne contribuent pas à l'augmentation du nombre d'occurrences de type t. Ce sont les variables dites perdues du point de vue de C_c.

Le deuxième cas (cas 2) concerne les variables dont les domaines sont réduits à un sous-ensemble du type t. Donc, ces variables contribueront nécessairement à l'augmentation du nombre d'occurrences de type t. Ce sont les variables dites gagnées de point de vue de C_c.

Le dernier cas (cas 3) concerne les variables dont les domaines contiennent des occurrences de type t et d'autres non. Ces variables sont dites indécises. En effet, selon le déroulement de la recherche, elles deviendront gagnées ou perdues du point de vue de C_c.

On obtient les paramètres $\bar{n}$, $\underline{n}$ *et* $\tilde{n}$ comme suit :

- $\underline{n} = \left|\{i = 1..k : \Theta(x_i) = \{1\}\}\right|$,
- $\bar{n} = \left|\{i = 1..k : \Theta(x_i) = \{1\}\}\right| + \left|(i = 1..k : \Theta(x_i) = \{0,1\}\}\right|$,
- $\tilde{n} = \alpha * (\bar{n} + \underline{n})$ *avec* : $\alpha \in]0,1[$ coefficient de prévision.

α définit la tendance d'instanciation des variables durant le déroulement de la recherche. Plus précisément, il indique la portion des variables indécises (cas3 équation 3.3) qui sont susceptibles de devenir participantes (cas2 équation 3.3) à la fin de la recherche. Par exemple, $\alpha = ½$ indique qu'on suppose que la moitie des variables indécises vont devenir participantes vers la fin de la recherche.

Reprenons l'exemple (3.2) :

Considérant que h_{card} est l'heuristique liée à la contrainte C_{c1}. Le tableau 3.2 montre les domaines des variables actuels.

D_{x_1} ne contient que des valeurs impaires, donc x_1 est participante $\Theta(x_1) = \{1\}$. D_{x_2} contient des valeurs paires et des valeurs impaires, donc x_2 est indécis $\Theta(x_2) = \{0,1\}$. x_3 n'est pas concernée par la contrainte C_{c1}. Alors que D_{x_4} ne contient que des valeurs impaires, donc x_4 est perdue $\Theta(x_4) = \{0\}$.

Tableau 3.2 : domaines des variables de l'exemple 3.2

Variable	x_1	x_2	x_3	x_4
Domaine	$\{1,3\}$	$\{1,2,3,4,a\}$	$\{1,2,3,b\}$	$\{2,4,a,b\}$
$\Theta(x_i)$	$\{1\}$	$\{0,1\}$	NA	$\{0\}$

D'où, $\underline{n} = 1$, $\bar{n} = 2$ *et* $\tilde{n} = (1+2) * ½ = 1.5$ *avec* $\alpha = ½$.

3.2.2.2 Mise à jour de l'état de l'heuristique

L'état de l'heuristique h_c S est une mesure du niveau de la satisfaction de C_c compte tenu de l'état actuel de la recherche. Nous proposons trois états pour l'heuristique Cardinalité, soit O^-, I et O^+. La figure 3.3 montre ces trois états.

Rappelons que :

- $\underline{n}$: le nombre d'occurrences effectivement gagnées,
- $\overline{n}$: le nombre maximal d'occurrences qu'on peut atteindre,
- $\tilde{n}$: moyenne prévisionnelle d'occurrences participantes acquises vers la fin de la recherche,
- $\overline{v}$ *et* $\underline{v}$: borne supérieure et inférieure de la contrainte C_c,
- $m = |Y|$: taille de l'ensemble des variables concernées.

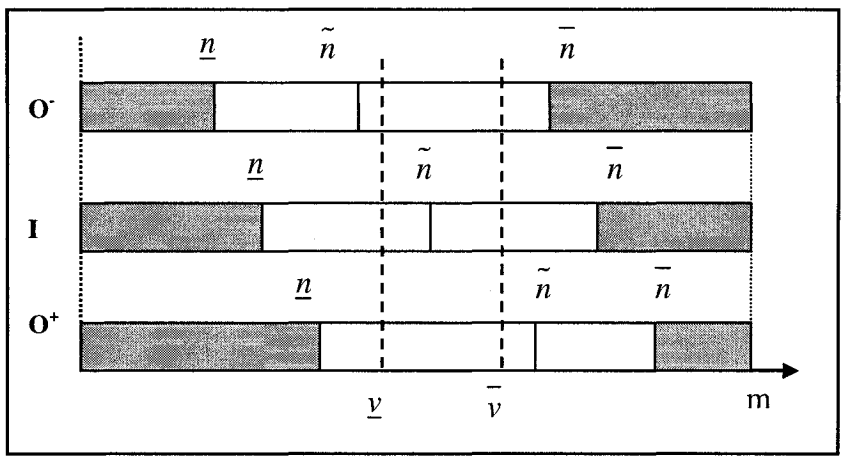

Figure 3.1 : états de l'heuristique Cardinalité avec $\alpha = 1/2$:

L'état O^- signifie que si la tendance d'instanciation des variables se maintient la contrainte C_c sera violée à cause de la borne minimale. Alors que l'état I indique que si la tendance se maintient la contrainte sera satisfaite. Par ailleurs, l'état O^+ annonce que, si la tendance se maintient, la contrainte C_c sera violée à cause de la borne maximale. Les heuristiques essayent constamment de rejoindre l'état I.

S est obtenu comme suit (voir la figure 3.3 ci-dessus) :

$$S = \begin{cases} O^- & si \quad \tilde{n} \le \underline{v} \\ I & si \quad \underline{v} < \tilde{n} < \overline{v} \\ O^+ & si \quad \tilde{n} \ge \overline{v} \end{cases} \tag{3.4}$$

Reprenons l'exemple (3.2) :

Considérant les domaines des variables donnés dans la section 3.2.2.1. Rappelons que $\overline{v} = 2$, $\underline{v} = 1$ *et* $\tilde{n} = 1.5$, on aura donc selon (3.4) l'état de l'heuristique $S_{h_{card}} = I$ puisque $\underline{v} < \tilde{n} < \overline{v}$.

3.2.2.3 Mise à jour des incitations à appliquer

Rappel et notations :

- A : l'ensemble des affectations, autrement dit les couples (variable, valeurs). Tel que :

  $$A = \bigcup_{i=1}^{m} \{x_i\} \times D_{X_i},\ A = \{a_1, a_2, .. a_r\}$$

  avec $a_i = (x_s, v_j)$ *tel que* $x_s \in V$ *et* $v_j \in D_{X_s}$

  a_i est l'affectation de la valeur v_j à la variable x_s,

- $A^0 = \{(x,v) \in A : v \notin t\}$, les affectations qui ne participent pas à l'augmentation du nombre d'occurrences du type concerné par la contrainte C_c.
- $A^1 = \{(x,v) \in A : v \in t\}$, les affectations qui participent à l'augmentation du nombre d'occurrences du type concerné par la contrainte C_c.
- A^1 *et* A^0 constituent une partition de A car $A^1 \cup A^0 = A$ *et* $A^1 \cap A^0 = \phi$.

Comme nous l'avons indiqué auparavant, l'état O^- signifie un risque de violation de la borne minimale. Alors que l'état O^+ annonce un risque de violation de la borne supérieure. Donc, nous devons favoriser les affectations qui participent à l'augmentation du nombre d'occurrences du type concerné par la contrainte C_c lorsque celle-ci est dans l'état O^- et les décourager lorsque celle-ci est dans l'état O^+. Les mises à jour des incitations sont appliquées seulement lors des changements d'état de l'heuristique.

Rappelons que chaque heuristique n'est réveillée que lorsqu'un événement survient dans les domaines des variables auxquelles elle est liée, donc seulement lorsque son état peut être affecté. Les incitations visent à garder l'heuristique dans l'état ***I***.

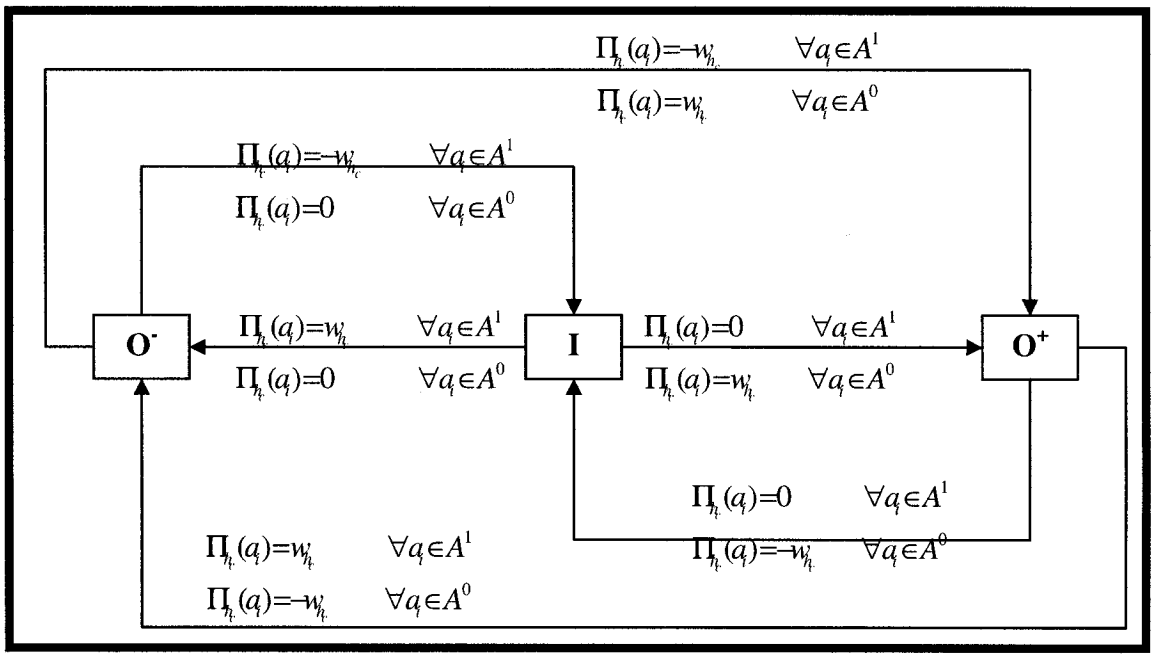

Figure 3.2 : Transitions et mises à jour des incitations pour l'heuristique Cardinalité

Nous allons noter S' l'état précédent de l'heuristique d'incitation, S son état actuel et $\Pi_{h_c}(a_i)$ l'incitation appliquée par celle-ci à l'affectation a_i. Rappelons que w_{h_c} est le poids de l'heuristique h_c. Six transitions sont possibles selon S' et S. La figure (3.4) ci-dessus résume les transitions possibles ainsi que les mises à jour des incitations effectuées lors de chaque transition.

Ci-dessous l'explication des six transitions d'états montrés dans la figure 3.4 :

- Premier cas $S' = I \; et \; S = O^+$: on attribue w_{h_c} aux affectations non participantes (A^0) pour éviter une violation de la borne maximale.
- Deuxième cas $S' = I \; et \; S = O^-$: on attribue w_{h_c} aux affectations participantes (A^1) pour éviter la violation de la borne minimale.
- Troisième cas $S' = O^+ \; et \; S = I$: on attribue $-w_{h_c}$ aux affectations non participantes (A^0) puisque l'heuristique est dans l'état d'équilibre ***I***.
- Quatrième cas $S' = O^+ \; et \; S = O^-$: on attribue w_{h_c} aux affectations participantes (A^1) et $-w_{h_c}$ aux affectations non participantes (A^0) pour rejoindre l'état d'équilibre ***I***.
- Cinquième cas $S' = O^- \; et \; S = I$: on attribue $-w_{h_c}$ aux affectations participantes (A^1) puisque l'heuristique a atteint l'état d'équilibre ***I***.
- Sixième cas $S' = O^- \; et \; S = O^+$: on attribue w_{h_c} aux affectations non participantes (A^0) et $-w_{h_c}$ aux affectations participantes (A^1) afin d'éviter la violation de la borne maximale.

Reprenons l'exemple (3.2) :

Considérons que l'état précédent est $S'_{h_{card}} = O^-$. Vu que l'état actuel est $S_{h_{card}} = I$ on est dans le cinquième cas ($S' = O^- \; et \; S = I$). Donc, on doit enlever les incitations aux assignations participantes (A^1). Rappelons que $w_{h_{card}}$ est la pondération (le poids) de l'heuristique h_{card} et que les domaines sont comme suit : $D_{x_1} = \{1,3\}$, $D_{x_2} = \{1,2,3,4,a\}$, $D_{x_3} = \{1,2,3,b\}$, $D_{x_4} = \{2,4,a,b\}$.

Tableau 3.3 : mise à jour des incitations par h_{card}

a_i	*Participantes* A^1	*Non-participantes* A^0	*Incitation*
$(x_1,1)$	✓	×	$-w_{h_c}$
$(x_1,3)$	✓	×	$-w_{h_c}$
$(x_2,1)$	✓	×	$-w_{h_c}$
$(x_2,2)$	×	✓	0
$(x_2,3)$	✓	×	$-w_{h_c}$
$(x_2,4)$	×	✓	0
$(x_3,1)$	×	✓	0
$(x_3,2)$	×	✓	0
$(x_3,3)$	×	✓	0
$(x_4,2)$	×	✓	0
$(x_4,4)$	×	✓	0

3.2.3 heuristique d'incitation pour la contrainte Demande

Dans cette section, nous commençons par présenter la contrainte **Demande** qui définit les services offerts aux clients. Puis, nous présentons l'heuristique d'incitation pour cette dernière.

3.2.3.1 La demande

Dans le modèle d'Hibiscus, une journée est divisée en intervalles appelés périodes. Un quart est constitué d'un ensemble de périodes contiguës. Ainsi, une période est caractérisée par l'ensemble des quarts qui la couvrent. Les périodes sont choisies minimales de telle sorte que les quarts peuvent êtres représentés en fonction des périodes.

La figure 3.1 montre un exemple de découpage d'une journée en période (P1, P2, P3 et P4) et en quarts (D8, D12, E1, E E2, S12 et S8).

La Demande (le service à assurer) est exprimée par période, par niveau de compétence du personnel, par jour. Par exemple, une contrainte de Demande pourrait être formulée comme suit : nous avons besoin de cinq personnes de niveau de compétence 2 pendant les périodes P3 tous les lundis.

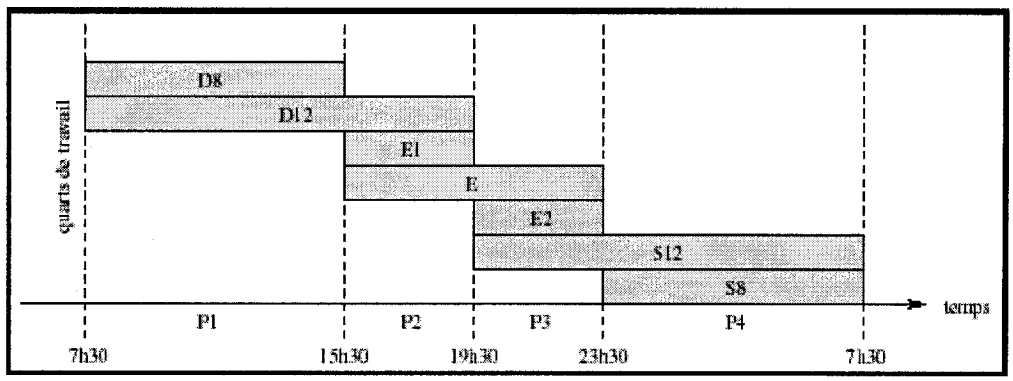

Figure 3.3 : Exemple de découpage d'une journée en périodes et en quarts. [1]

Dans Hibiscus, la Cardinalité est exprimée sous forme d'un triplet (cible, déficit, excès). La cible définit l'objectif optimal visé, le déficit est l'écart accepté en dessous de la cible, alors que l'excès est l'écart accepté en dessus de la cible. Voir ci-dessous la définition de la contrainte Demande.

Définition 3.4 : Contrainte Demande.

Une contrainte Demande C_{de} est définie par $C_{de}(I,T,O,D,E,J)$ tel que :

- $I=\{i_1,i_2,..i_k\}$: sous ensemble des infirmières concernées. $I \subset P$ avec P l'ensemble du personnel,
- T : sous-ensemble de quarts caractérisant la période de demande voulue, tel que $T \subset Q$ et $Q=\{q_1,\ q_2,\ ..\ q_l\}$ l'ensemble de quarts,
- O : la cible (objectif), soit le nombre d'occurrences du type T voulu,
- D : le déficit permis vis-à-vis de la cible,
- E : l'excès permis vis-à-vis de la cible,
- $J=\{j_1,j_2,....,j_h\}$: l'ensemble des jours concernés.

Le tableau 3.1 ci-dessous présente un exemple de spécification d'une contrainte de demande. Cette contrainte impose que le nombre de périodes P3 (P3 est couverte par les quarts E, E2, S12) travaillées tous les lundis par le personnel de niveau 2 soit 6. Cependant, un excès et un déficit de 1 sont permis. Donc, le nombre de périodes doit être compris entre 5 (6-1) et 7 (6+1).

Tableau 3.4 : Spécification d'une contrainte demande.

I	T	O	D	E	J
2	E, E2, S12	6	1	1	lundi

3.2.3.2 Heuristique d'incitation pour la contrainte Demande

Dans cette section, nous montrons l'application de l'heuristique d'incitation Cardinalité à la contrainte Demande.

Supposons une contrainte Demande $C_{de}(I_{de}, T_{de}, O_{de}, D_{de}, E_{de}, J_{de})$, telle que I_{de} le sous-ensemble des infirmières concerné, T_{de} le sous-ensemble de quarts caractérisant la période de demande voulue, O_{de} la cible, D_{de} le déficit permis, E_{de} l'excès permis et J_{de} le l'ensemble de jours concerné.

Rappelons qu'une heuristique d'incitation h_c est définie par un couple (C_c, w_{h_c}) (définition 3.3) où C_c est la contrainte de Cardinalité à laquelle l'heuristique h_c est liée et w_{h_c} le poids de l'heuristique h_c. Par ailleurs, une contrainte de Cardinalité C_c est définie par un quadruplet $(Y, t, \bar{v}, \underline{v})$ (définition 3.2) où t est le type concerné, Y est l'ensemble de variables concernées, $\bar{v}$ la borne maximale et $\underline{v}$ la borne minimale.

$C_{c_{de}}$ est dite la contrainte de Cardinalité correspondante à la contrainte C_{de}. C_{de} est défini comme suit $C_{c_{de}}(Y, t, \bar{v}, \underline{v})$ tel que :

- $Y = I_{de} \times J_{de}$: les variables concernées,
- $t = T_{de}$: le sous-ensemble de type concerné,
- $\bar{v} = O_{de} + E_{de}$: la borne maximale,
- $\underline{v} = O_{de} - D_{de}$: la borne minimale.

On définit alors l'heuristique Cardinalité h_{de} liée à la contrainte Demande C_{de} comme suit $h_{de}(C_{c_{de}}, w_{de})$.

3.2.4 Quarts non couverts

Dans cette section, nous présentons le mécanisme qui a permis de résoudre le problème d'instanciation tardive des quarts de travails détecté lors des tests préliminaires.

Considérons les notations suivantes :

- $Q = \{q_1, q_2, .., q_r\}$ l'ensemble des quarts,
- $t_{h^i_{de}} \subset Q$, $t_{h^i_{de}} = \{q_i, .., q_j\}$: l'ensemble des quarts contrôlés par l'heuristique h^i_{de},
- $H_{de} = \{h^1_{de}, h^2_{de}, .., h^n_{de}\}$ l'ensemble des heuristiques Cardinalité actives,
- $T_{H_{de}} = \bigcup_{i=1}^{i=n} t_{h^i_{de}}$ l'ensemble des quarts contrôlés,
- $\overline{T}_{H_{de}} = Q - T_{H_{de}}$ l'ensemble des quarts libres (non contrôlés par une heuristique Cardinalité). Cet ensemble contient au moins les quarts *Vacances* et *Repos*,
- C^i_{de} la contrainte Demande à laquelle l'heuristique h^i_{de} est liée.

Chaque heuristique h^i_{de} effectue principalement deux types d'incitations :

> Cas 1. Lorsque le risque de violation de la contrainte C^i_{de} concerne la borne minimale. Alors, l'heuristique h^i_{de} incite les quarts participants.
>
> Cas 2. Lorsque le risque de violation de la contrainte C^i_{de} concerne la borne maximale. Alors, l'heuristique h^i_{de} incite les quarts non participants.

Le problème est situé dans le 'Cas 2.' Vu que les quarts $\overline{T}_{H_{de}}$ ne sont contrôlés par aucune heuristique, ils se retrouvent alors dans tous les ensembles non participants $\overline{t}_{h^i_{de}}$ des heuristiques. Dans le cas où plusieurs heuristiques se retrouvent dans le 'Cas 2.', les $\overline{T}_{H_{de}}$ bénéficieront de plusieurs incitations à la fois. Donc, le système va favoriser leurs instanciations, d'où l'instanciation tardive des quarts contrôlés $T_{H_{de}}$.

Souvent, l'ajout de contraintes redondantes permet l'amélioration du filtrage en couvrant des aspects de dépendance entre variables non couverts par les contraintes d'origine. Le principe derrière la solution au problème d'instanciation tardive des quarts de travails est inspiré de celui des contraintes redondantes.

Vu que la demande n'est exprimée que sur un sous-ensemble de périodes, seule l'instanciation d'un sous-ensemble de quarts est contrôlée par les heuristiques. La solution alors est d'ajouter une heuristique, nommée heuristique complémentaire, qui couvrira les quarts libres. L'implémentation de cette solution permet une amélioration intéressante des performances. L'annexe 1 présente un exemple d'instanciation qui permet de visualiser le problème ainsi que la solution proposée.

3.3 Contrainte Cardinalité pondérée

Le deuxième type de contrainte que nous traitons est la Cardinalité pondérée. Dans Hibiscus la contrainte d'équilibrage des types des quarts, nommée aussi contrainte Équilibrage, est une contrainte de Cardinalité pondérée. Cette contrainte est souvent difficile à satisfaire.

3.3.1 Cardinalité pondérée

Dans cette section, nous présentons la contrainte de Cardinalité pondérée et les heuristiques d'incitation proposées pour cette dernière. Ensuite, nous présentons la contrainte Équilibrage ainsi que l'application des heuristiques proposées à celle-ci.

Dans notre cas la contrainte de Cardinalité pondérée impose des pondérations, sous forme de pourcentages, sur les types d'occurrences concernés. Ci-dessous une définition formelle de cette contrainte. (Définition 3.5)

Définition 3.5 : Contrainte de Cardinalité pondérée.

Une contrainte de Cardinalité pondérée C_{cp} est définie par le triplet (X, T, L) tel que :

- $X = \{x_i, x_j, .., x_n\}$ l'ensemble des variables concernées,
- $T = \{t_1, t_2, .., t_m\}$ vecteur des types concernés,
- $L = \{l_i, l_j, .., l_m\}$ vecteur des limites minimales et maximales, tel que :

$$l_k = (\underline{r}_k, \overline{r}_k) \; avec \; \underline{r}_k, \overline{r}_k \in [01] \; et \; \sum_{k=1}^{m} \underline{r}_k \leq 1$$

- $\underline{r}_k, \overline{r}_k$ sont respectivement la limite minimale et maximale des occurrences de type t_k.

La contrainte C_{cp} impose que $\underline{r}_k \tau \leq \tau_k \leq \overline{r}_k \tau$. où τ_k désigne le nombre d'occurrences instanciées de type t_k et $\tau = \sum_{k=1}^{j=m} \tau_k$ le nombre total d'occurrences instanciées de type T. Une contrainte de Cardinalité pondérée C_{cp} est un ensemble de contraintes de Cardinalité tel que : $C_{cp} = \{C_{c_k}(X, \tau_k, \overline{r_k} * \tau, \underline{r_k} * \tau) | 1 \leq k \leq m\}$. Par conséquence, nous allons nous inspirer de l'heuristique Cardinalité dans la conception de l'heuristique de Cardinalité pondérée.

Exemple 3.3 :

Considérant les données suivantes :

- $X = \{x_1, x_2, .., x_{10}\}$: ensembles des variables.
- $V = \{1, 2, 3, 4, a, b\}$: vecteur des valeurs.
- $D_{x_1} = D_{x_2} = .. = D_{x_{10}} = \{1, 2, 3, 4, a, b\}$: domaines des variables.
- $T = \{t_1, t_2\}$ *avec* $t_1 = \{1, 3\}$ *et* $t_2 = \{2, 4\}$: ensemble des types.
- $L = \{(50\%, 70\%), (30\%, 50\%)\}$: limites minimales et maximales.

Une contrainte de Cardinalité pondérée $C_{cp}(X, \mathrm{T}, L)$ impose qu'entre 50 % et 70 % des variables doivent avoir des valeurs impaires alors qu'entre 30 % et 50 % des doivent avoir des valeurs paires.

3.3.2 Heuristique d'incitation pour la contrainte Cardinalité pondérée

Nous avons conçu trois heuristiques pour la contrainte de Cardinalité pondérée. Les trois heuristiques sont inspirées de celle utilisée pour la contrainte de Cardinalité (section 3.2). La première heuristique effectue le suivi des types conjointement, alors que la deuxième et la troisième effectuent le suivi des types séparément. Dans cette section, nous présentons ces trois heuristiques.

3.3.2.1 Heuristique d'incitation Max_Déficit

La première heuristique est l'heuristique Max_Déficit. Elle effectue le suivi de tous les types conjointement. Elle incite seulement le type le plus déficient, soit celui dont l'écart entre la prévision et la borne minimale de sa Cardinalité est le plus grand.

Définition 3.6 : Heuristique d'incitation Max_Déficit.

Une heuristique d'incitation Max_Déficit $h_{MaxDef}(C_{cp},\ w_{h_{Max\_Def}})$ est définie tel que :

- C_{cp} : La contrainte de Cardinalité pondérée à laquelle l'heuristique h_{MaxDef} est liée.
- $w_{h_{Max\_Def}}$: Le poids de l'heuristique h_{MaxDef}.

Comme pour l'heuristique de Cardinalité (section 3.2.1), à chaque événement dans le domaine d'une variable de *V*, l'heuristique effectue la mise à jour des paramètres, de l'état de l'heuristique et des incitations à appliquer.

Reprenons l'exemple 3.3 :

La contrainte de cardinalité pondérée est définit comme suite $C_{CP} = \{C_{c_1}(X_1, t_1, 7, 5), C_{c_2}(X_2, t_2, 5, 3)\}$ *avec* $X_1 = X_2 = X$ Il. Il reste à définir $w_{h_{MD}}$ le poids de l'heuristique selon l'importance de la contrainte C_{CP} dans le problème. Alors l'heuristique d'incitation liée à C_{CP} est $h_{MD}(C_{CP},\ w_{h_{MD}})$.

a. Mise à jour des paramètres

La première opération effectuée par l'heuristique h_{MaxDef} est la mise à jour des vecteurs $\overline{N}, \underline{N}$ *et* $\tilde{N}$, telle que :

- $\underline{N} = \{\underline{n}_k \mid 1 \le k \le m\}$ *avec* $\underline{n}_k = |i = 1..s_k : \Theta_k(x_i) = \{1\}|$ *et* $s_k = |X_k|$: Le vecteur des occurrences participantes gagnées par type, avec $m = |T|$.
- $\overline{N} = \{\overline{n}_k \mid 1 \le k \le m\}$ *avec* $\overline{n}_k = \underline{n}_k + |i = 1..s_k : \Theta_k(x_i) = \{0,1\}|$ *et* $s_k = |X_k|$: Le vecteur des occurrences maximales participantes maximales par type.
- $\tilde{N} = \{\tilde{n}_k \mid 1 \le k \le m\}$ *avec* $\tilde{n} = \alpha * (\overline{n_k} + \underline{n}_k)$, $\alpha \in]0,1[$ *coefficient de prévision* Le vecteur des prévisions par type.

Reprenons l'exemple 3.3 :

Rappelons que les types sont définis comme suite $T = \{t_1, t_2\}$ *avec* $t_1 = \{1,3\}$ *et* $t_2 = \{2,4\}$. Θ_i est la fonction qui définit le type de contribution des variables dans la satisfaction de la contrainte C_{c_i}. Θ_i est défini exactement de la même façon que Θ (section 3.2.2.1). Le tableau 3.5 montre les domaines et les contributions de chaque variable.

Tableau 3.5 : Contribution des affectations (exemple 3.3)

variables	D_{x_i}	$\Theta_1(x_i)$	$\Theta_2(x_i)$
x_1	$D_{x_1}=\{1\}$	$\{1\}$	$\{0\}$
x_2	$D_{x_2}=\{1,2,a\}$	$\{0,1\}$	$\{0,1\}$
x_3	$D_{x_3}=\{1,3\}$	$\{1\}$	$\{0\}$
x_4	$D_{x_4}=\{2,4\}$	$\{0\}$	$\{1\}$
x_5	$D_{x_5}=\{1,4,b\}$	$\{0,1\}$	$\{0,1\}$
x_6	$D_{x_6}=\{1\}$	$\{1\}$	$\{0\}$
x_7	$D_{x_7}=\{1,2,a,b\}$	$\{0,1\}$	$\{0,1\}$
x_8	$D_{x_8}=\{3\}$	$\{1\}$	$\{0\}$
x_9	$D_{x_9}=\{1\}$	$\{1\}$	$\{0\}$
x_{10}	$D_{x_{10}}=\{1,3\}$	$\{1\}$	$\{0\}$

Donc, $\underline{N}=\{\underline{n}_1,\underline{n}_2\}$ *avec* $\underline{n}_1=6$ *et* $\underline{n}_2=1$, $\overline{N}=\{\overline{n}_1,\overline{n}_2\}$ *avec* $\overline{n}_1=9$ *et* $\overline{n}_2=4$ et $\tilde{N}=\{\tilde{n}_1,\tilde{n}_2\}$ *avec* $\tilde{n}_1=7.5$ *et* $\tilde{n}_2=2.5$.

b. Mise à jour de l'état de l'heuristique.

La deuxième opération est la mise à jour de l'état de l'heuristique h_{MaxDef}. L'état de l'heuristique h_{MaxDef} indique le type incité par celle-ci $S = \{t_1, t_2, .., t_n, Aucun\}$ ($S = Aucun$ indique qu'aucun type n'est incité).

L'heuristique incite le type le plus déficient, soit celui ayant l'écart $\Delta_k = \underline{r}_k * \tau - \tilde{n}_k$ le plus grand. Avec $\underline{r}_k * \tau$ le nombre de variable instanciées de type k et $\tilde{n}_k$ est la prévision.

L'état de l'heuristique S est obtenu comme suit :

$$S = \begin{cases} t_k & si\ \Delta_k > 0\ et\ \Delta_k = Max(\Delta_i)\ \forall\ 1 \le i \le m \\ Aucun & si\ \Delta_i \le 0 \qquad\qquad \forall\ 1 \le i \le m \end{cases} \tag{3.5}$$

Reprenons l'exemple 3.3 :

Notons $S_{h_{MD}}$ l'état de l'heuristique h_{MD}. On a les déficits $\Delta_1 = \underline{r}_1 - \tilde{n}_1 = 5 - 7.5 = -2.5$ et $\Delta_2 = \underline{r}_2 - \tilde{n}_2 = 3 - 2.5 = 0.5$. Donc, l'état de l'heuristique est $S_{h_{MD}} = t_2$.

c. Mise à jour des incitations à appliquer.

Considérons les notations suivantes :

- $A^k = \{(x,v) \in A : v \in t_k\}$, : l'ensemble des affectations participantes à l'augmentation des occurrences du type t_k,
- $S' = t_{k'}$ l'état précédent de l'heuristique,
- $S = t_k$ le nouvel état de l'heuristique,
- $\Pi_{h_{MaxDef}}(a_i)$ L'incitation appliquée par l'heuristique h_{MaxDef} à l'affectation a_i.

Comme pour l'heuristique Cardinalité, les incitations ne sont appliquées que lors des transitions entre états. À chaque changement d'état, l'heuristique incite (attribuer $w_{h_{MaxDef}}$) les affectation de type k et enlève les incitations aux affectations de type k' (attribuer $-w_{h_{MaxDef}}$) :

$$\Pi_{h_{MaxDef}}(a_i)\begin{cases} w_{h_{MaxDef}} & si\ S = t_k \quad \forall\ a_i \in A^k \\ -w_{h_{MaxDef}} & si\ S' = t_{k'} \quad \forall\ a_i \in A^{k'} \\ 0 & \text{sinon} \end{cases} \qquad (3.6)$$

Reprenons l'exemple 3.3 :

Puisque l'état de l'heuristique est $S_{h_{MD}} = t_2$, seules les affectations ayant une valeur paire seront incitées. Le tableau 3.6 montre les affectations incitées par h_{MD}.

Tableau 3.6 : Les affectations incitées par h_{MD} (exemple 3.3)

Affectations	Incitation
$(x_2, 2)$	$w_{h_{MD}}$
$(x_4, 2)$	$w_{h_{MD}}$
$(x_4, 4)$	$w_{h_{MD}}$
$(x_5, 4)$	$w_{h_{MD}}$
$(x_7, 2)$	$w_{h_{MD}}$

3.3.2.2 Heuristique d'incitation Indépendante

L'heuristique indépendante effectue les incitations des types indépendamment les uns des autres, d'où son nom. Les poids d'incitation des types sont égaux.

Définition 3.7 : heuristique d'incitation Indépendante

Une heuristique d'incitation Indépendante h_{Ind} est définie par le couple $(H_c, w_{h_{Ind}})$ avec :

- $w_{h_{Ind}}$: le poids de l'heuristique h_{Ind}.
- $H_c = \{h_{c_1}, h_{c_2}, .., h_{c_m}\}$, *avec* $h_{c_k}(C_{c_k}, w^k_{h_{Pond}})$ *et* $C_{c_k}(X, \tau_k, \overline{r_k}, \underline{r_k})$ $\forall$ $1 \le k \le m$.

  l'ensemble des heuristiques Cardinalité correspondant à h_{Pond}.

La définition 3.7 montre que l'heuristique Indépendante est constituée d'un ensemble d'heuristiques Cardinalité $H_c = \{h_{c_1}, h_{c_2}, .., h_{c_m}\}$. Les heuristiques d'incitation h_{c_k} fonctionnent indépendamment les unes des autres. Elles fonctionnent exactement de la même façon que dans la section (3.2.3).

Reprenons l'exemple 3.3 :

L'heuristique d'incitation Indépendante $h_I(H_c, w_{h_I})$ liée à C_{cp} est définie tel que :

$$H_c = \{h_{c_1}, h_{c_2}\} \text{ avec } h_{c_1}(C_{c_1}, w_{h_I}) \text{ et } h_{c_2}(C_{c_2}, w_{h_I}) \text{ telque :}$$
$$C_{c_1}(X_1, t_1, 7, 5) \text{ et } C_{c_2}(X_2, t_2, 5, 3)$$
$$\text{Rappel} : C_{cp} = \{C_{c_1}(X_1, t_1, 7, 5), C_{c_2}(X_2, t_2, 5, 3)\} \text{ et } X_1 = X_2 = X$$

3.3.2.3 Heuristique d'incitation Pondérée

La troisième heuristique, appelée heuristiques Pondérée, effectue les incitations des types indépendamment les uns des autres, comme dans le cas de l'heuristique Indépendante. Mais, elle utilise des poids par type distincts. Ci-dessous la définition de cette heuristique.

Définition 3.8 : heuristique d'incitation Pondérée

Une heuristique d'incitation Pondérée h_{Pond} est définie par le couple $(H_c, W_{h_{Pond}})$ avec :

- $W_{h_{Pond}} = \left\{ w^1_{h_{Pond}}, w^2_{h_{Pond}}, ..., w^m_{h_{Pond}} \right\}$ *tel que* $w^k_{h_{Pond}} = ((\overline{r_k} + \underline{r_k}) \div 2) * w_{h_{Pond}} \; \forall \; 1 \le k \le m$ le vecteur des poids.
- $H_c = \left\{ h_{c_1}, h_{c_2}, ..., h_{c_m} \right\}$ *avec* $h_{c_k}(C_{c_k}, w^k_{h_{Pond}})$ *et* $C_{c_k}(X, \tau_k, \overline{r_k}, \underline{r_k}) \; \forall \; 1 \le k \le m$ l'ensemble des heuristiques Cardinalité, correspondant à h_{Pond}.

Là aussi, l'heuristique Pondérée est composée de plusieurs heuristiques Cardinalité. La seule différence entre l'heuristique Indépendante et l'heuristique Pondérée est au niveau des poids d'incitation utilisés. Dans le cas de l'heuristique Indépendante toutes les heuristiques h_{c_k} ont le même poids. Alors que pour l'heuristique Pondérée, le poids de chaque heuristique h_{c_k} dépend des bornes de sa Cardinalité.

Les heuristiques d'incitation h_{c_k} composant h_{Pond} fonctionnent indépendamment les unes des autres. Ces heuristiques fonctionnent exactement de la même façon que dans la section (3.2.3).

Reprenons l'exemple 3.3 :

L'heuristique d'incitation Indépendante $h_p(H_c, W_{h_p})$ liée à C_{cp} est définie par tel que :

$$H_c = \{h_{c_1}, h_{c_2}\}$$
$$h_{c_1}(C_{c_1},\ w^1_{h_p})\ \ avec\ C_{c_1}(X_1, t_1, 7, 5),\ w^1_{h_p} = w_{h_p} * ({}^{(7+5)}/_{2}) = 6 * w_{h_p}$$
$$h_{c_2}(C_{c_2},\ w^2_{h_p})\ avec\ C_{c_2}(X_2, t_2, 5, 3),\ w^2_{h_p} = w_{h_p} * ({}^{(5+3)}/_{2}) = 4 * w_{h_p}$$
$$Rappel : C_{cp} = \{C_{c_1}(X_1, t_1, 7, 5), C_{c_2}(X_2, t_2, 5, 3)\}\ \ et\ \ X_1 = X_2 = X$$

3.3.3 heuristique d'incitation Cardinalité pondérée pour la contrainte Équilibrage

Dans cette section, nous commençons par donner une définition de la contrainte Équilibrage. Puis, nous présentons l'application des trois heuristiques d'incitation étudiées dans la section précédente à la contrainte Équilibrage.

3.3.3.1 Contrainte Équilibrage

La contrainte Équilibrage fait partie de la classe des contraintes ergonomiques. Cette classe contient toute règle visant à maintenir un bon niveau de qualité des horaires

individuels. Les règles ergonomiques sont dictées par la réglementation, les conventions, les choix des membres du personnel, etc. La contrainte Équilibrage impose des limites minimales et maximales sur les pourcentages de chaque type des quarts travaillés. Dans notre cas on distingue trois types de quarts, soit le jour D, le soir E et la nuit N.

Définition 3.9 : Contrainte **Équilibrage.**

Une contrainte Équilibrage $C_{éq}$ est définie par le triplet (T, i, L) tel que :

- i le membre du personnel concerné.
- $T = \{t_1, t_2, t_3\}$ l'ensemble des types des quarts (1 : jour, 2 : soir, 3 : nuit).
- $L = \{l_1, l_2, l_3\}$ l'ensemble des limites min et max de chaque type de quart. Avec :

$$l_k = \{\underline{r_k}, \overline{r_k}\},\ (\underline{r_k}, \overline{r_k}) \in [0\ 1]\ et\ \sum_j^s \underline{r_j} \leq 1.$$

- $\underline{r_k}, \overline{r_k}$ respectivement la limite minimale et maximale (en pourcentage) du nombre de quarts travaillés de type k.

Tableau 3.7 : données spécifiant deux contraintes d'équilibrage.

i	Jour (1)		Soir (2)		Nuit (3)	
	$\underline{r_1}$	$\overline{r_1}$	$\underline{r_2}$	$\overline{r_2}$	$\underline{r_3}$	$\overline{r_3}$
1	40	50	20	40	10	30
2	0	0	40	60	40	60

Le tableau 3.7 donne les spécifications de deux contraintes Équilibrage. La première ligne du tableau 3.7 concerne une contrainte appliquée à l'infirmière numéro 1. Cette contrainte impose qu'entre 40 % et 50 % des quarts travaillés doivent être du jour, qu'entre 20 % et 40 % des quarts travaillés doivent être du soir et qu'entre 10 % et 30 % des quarts travaillés doivent être de la nuit.

3.3.3.2 Heuristiques d'incitation pour la contrainte Équilibrage

Dans cette section, nous présentons l'application des heuristiques Max_Déficit, Indépendante et Pondérée à la contrainte Équilibrage.

Soit une contrainte Équilibrage $C_{éq}(T,i,L)$ telle que :

- i : le membre du personnel concerné,
- $T=\{t_1,t_2,t_3\}$: l'ensemble des types de quart,
- $L=\{l_1,l_2,l_3\}$ *avec* $l_1=\{\overline{r_1},\underline{r_1}\},l_2=\{\overline{r_2},\underline{r_2}\}$ *et* $l_3=\{\overline{r_3},\underline{r_3}\}$: l'ensemble des limites minimales et maximales de chaque type.

La contrainte de Cardinalité pondérée $C_{cp_{éq}}$ correspondante à $C_{éq}$ tel que :

- $C_{cp_{éq}}=\{C_{c_1},C_{c_2},C_{c_3}\}$ *avec* $C_{c_1}(X,t_1,\overline{r_1},\underline{r_1})$, $C_{c_2}(X,t_2,\overline{r_2},\underline{r_2})$ *et* $C_{c_3}(X,t_3,\overline{r_3},\underline{r_3})$

- $X=\{X_{i,j}\,|\,j\in H\}$: l'ensemble des variables.
- H : l'horizon de planification concerné par la contrainte $C_{éq}$.

Cas 1. Heuristique d'incitation Max_Déficit

L'heuristique Max_Déficit liée à la contrainte Équilibrage $C_{éq}(T,i,L)$ est définie comme suit $h_{Max\_Def}(C_{cp_{éq}}, w_{éq})$ tel que :

- $w_{éq}$: Le poids de l'heuristique.
- $C_{cp_{éq}} = \{C_{c_1}, C_{c_2}, C_{c_3}\}$ contrainte de Cardinalité pondérée correspondante à la contrainte Équilibrage $C_{éq}$.

La figure 3.4 montre les transitions et les mises à jour des incitations appliquées, lors des transitions, par l'heuristique Max_Déficit appliquée à la contrainte Équilibrage.

Cas 2. Heuristique d'incitation Indépendante

L'heuristique d'incitation Indépendante liée à la contrainte Équilibrage $C_{éq}(T,i,L)$ est définie comme suit $h_{Ind}(H_c, w_{éq})$ tel que :

- $w_{éq}$: Le poids de l'heuristique.
- $H_c = \{h_{c_1}, h_{c_2}, h_{c_3}\}$ L'ensemble des heuristiques Cardinalité tel que :

  $h_{c_k}(C_{c_k}, w_{éq}) \ \forall \ 1 \le k \le 3$

  avec $C_{c_1}(X, t_1, \overline{r_1}, \underline{r_1})$, $C_{c_2}(X, t_2, \overline{r_2}, \underline{r_2})$ *et* $C_{c_3}(X, t_3, \overline{r_3}, \underline{r_3})$

cas 3. Heuristique d'incitation Pondérée

L'heuristique d'incitation Pondérée liée à la contrainte Équilibrage $C_{éq}(T,i,L)$ est définie comme suit $h_{Pond}(H_c, w_{éq})$ tel que :

- $w_{éq}$: Le poids de l'heuristique.
- $H_c = \{h_{c_1}, h_{c_2}, h_{c_3}\}$ l'ensemble des heuristiques d'incitation Cardinalité tel que :

$$h_{c_k}(C_{c_k},\ w^k_{h_{Pond}}) \text{ avec } C_{c_1}(X,t_1,\overline{r_1},\underline{r_1}),\ C_{c_2}(X,t_2,\overline{r_2},\underline{r_2}) \text{ et } C_{c_3}(X,t_3,\overline{r_3},\underline{r_3})$$

$$w^k_{h_{Pond}} = ((\overline{r_k} + \underline{r_k}) \div 2) * w_{éq} \qquad \forall\ 1 \le k \le 3$$

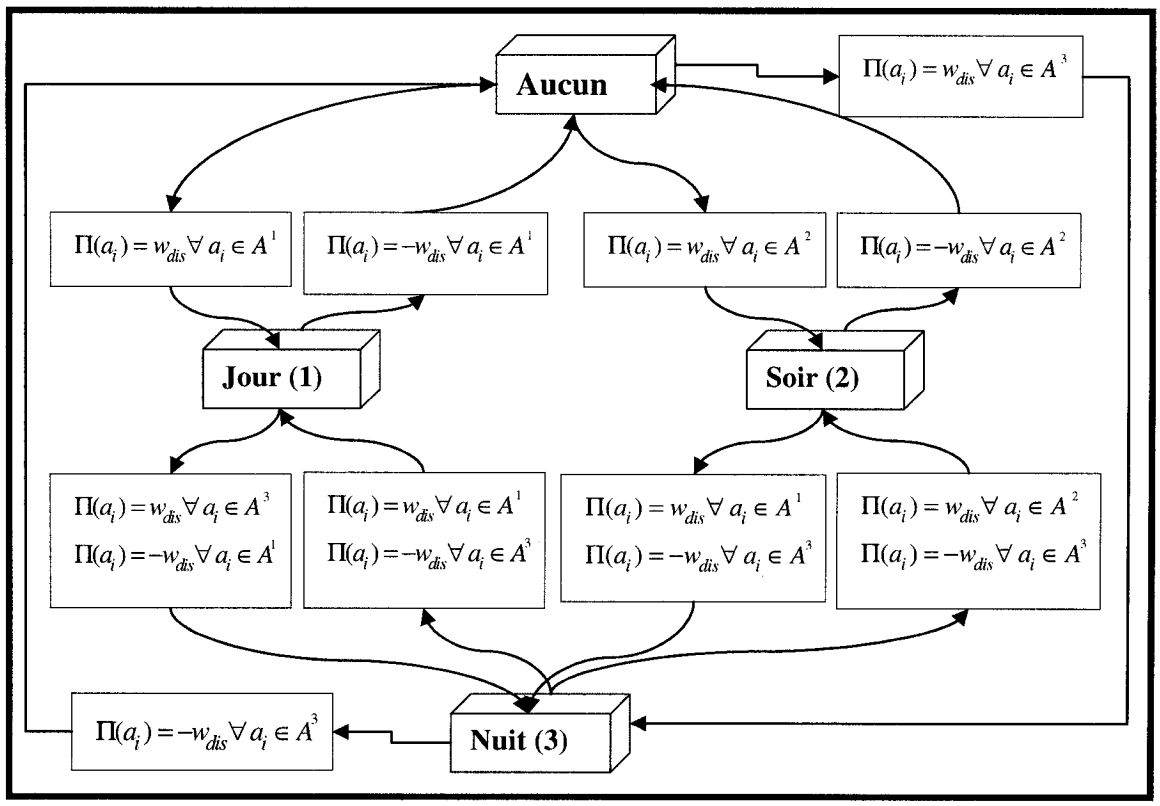

Figure 3.4 : Transitions et mise à jour des incitations pour l'heuristique Max_Déficit

Chapitre4 : Validation expérimentale

Dans ce chapitre, nous présentons les jeux de données choisis pour les tests, le plan d'expérimentation retenu pour la validation des heuristiques proposées ainsi que les résultats obtenus. Finalement, une discussion des résultats obtenus clôt ce chapitre.

4.1 Description des données

Dans cette section, nous donnons la description des jeux de données retenus pour les tests. Les descriptions des jeux de données ne contiennent que les détails jugés nécessaires pour une bonne mise en contexte.

Nous avons retenu quatre jeux de données. Tous ces jeux concernent des unités de soins infirmiers. Ces jeux sont :

- Centre des naissances de l'Hôpital Royal Victoria, nommé BC.
- Unité de pédiatrie de l'Hôpital Royal Victoria, nommé CHILD.
- Unité de dialyse de l'Hôpital Royal Victoria, nommé DIA.
- Unité de soins de la salle d'urgence de l'Hôpital Général de Montréal nommé ERMGH.

Pour chaque jeu de données, nous présentons :

- Les informations générales (taille du problème) : l'effectif, l'horizon, le nombre de périodes et le nombre des quarts. Les niveaux de compétences ne seront pas pris en compte dans ce travail de recherche,

- La décomposition (périodes, quarts) : Hibiscus décompose les 24 heures en périodes et en quarts,

- La demande : le service à assurer. Spécifier par jour, ou sous-ensemble de jours, le nombre de périodes voulues. La demande est exprimée sous forme d'un triplet (*cible, déficit, excès*). Les limites minimales et maximales sont obtenues comme suit : $\underline{v} = cible - déficit$ et $\overline{v} = cible + excès$,

- Les disponibilités : on spécifie les vacances, les préaffectations, les aversions et les quarts possibles pour chaque membre du personnel,

- Équilibrage : détermine les pourcentages de chaque type de quarts (trois types de quarts sont définis Jour, Soir et Nuit),

- La charge de travail : on indique la charge de travail en nombre d'heures (ou de quarts) par sous-ensemble de jours par membre du personnel,

- Les autres contraintes : Principalement, on y trouve les contraintes de la classe ergonomique (ERG) et équité (FAI), soit celles qui concernent la qualité des horaires. L'annexe 2 contient une description de toutes les contraintes.

4.1.1 Centre des naissances de l'Hôpital Royal Victoria (BC)

4.1.1.1 Informations générales

Tableau 4.1 : Description des informations générales (jeu BC)

Nom de fichier	BC_S01_N54.data (data2)
Effectif	54 infirmières
Horizon	6 semaines
Nombre de périodes	5
Nombre de quarts	12

4.1.1.2 Décomposition d'une journée

Périodes

Tableau 4.2 : Description des périodes (jeu BC)

Numéro de période	Début	Fin
1	07 : 30	11 : 30
2	11 : 30	15 : 30
3	15 : 30	19 : 30
4	19 : 30	23 : 30
5	23 : 30	07 : 30

Les quarts

Tableau 4.3 : Description des quarts (jeu BC)

Sigle	Périodes couvertes	Type
D	1, 2	Jour
SH	1, 2	Jour
DP	1, 2	Jour
D12	1, 2, 3	Jour
E1	3	Soir
E2	4	Soir
E	3, 4	Soir
EP	3, 4	Soir
S12	4, 5	Nuit
N	5	Nuit
S	5	Nuit
NP	5	Nuit

4.1.1.3 La demande

Tableau 4.4 : Description de la demande (jeu BC)

Jour	Période	Min($\underline{v}$)	Cible	Max($\bar{v}$)
Tous	1	5	9	17
Tous	2	5	9	17
Tous	3	5	9	17
Tous	4	5	9	17
Tous	5	5	9	17

4.1.1.4 Disponibilité

- Cinq semaines de vacances sont attribuées durant cet horizon.
- Quatre infirmières travaillent seulement les samedis et les dimanches.
- Six infirmières ont quelques préaffectations par semaine.
- Les quarts possibles sont indiqués dans la colonne (quarts possibles) du tableau 4.5 ci-dessous.

4.1.1.5 Équilibrage

Tableau 4.5 : Contraintes Équilibrage (jeu BC)

Équilibrage (*Matin, Soir, Nuit*)	Nombre de personnes	Quarts possibles
100 %, 0 %, 0 %	7	D, D12
0 %, 100 %, 0 %	6	E
0 %, 0 %, 100 %	8	N, S12
50 %, 50 %, 0 %	21	D, D12, E
50 %, 0 %, 50 %	7	D, D12, N, S12
0 %, 50 %, 50 %	3	E, N, S12
33 %, 33 %, 33 %	1	D, D12, E, N, S12
33 %, 66 %, 0 %	1	D, D12, E

4.1.1.6 Charge de travail

Pour ce jeu de données, la charge de travail est exprimée par période de deux semaines :

- 17 infirmières sont à temps complet, soit environ 80 heures de travail.
- 37 infirmières ont une charge de travail variant entre 24 et 64 heures.

4.1.1.7 Autres contraintes

- Succession des quarts de travails (*ConsecutiveWorkShifts*).
- Quart du jour suivant (*NextDayShifts*).
- Succession des quarts par type (*ConsecutiveShifts*).
- Pas de quarts isolés (*NoIsolatedShift*).
- Pas de quarts de repos isolés (*NoIsolatedOffShift*).
- Affectations prédéterminées (*PreAssignments*).
- Affectations interdites (*ForbiddenAssignments*).
- Fin de semaines consécutives (*ConsecutiveWeekends*).
- Pas de fin de semaine brisée (*NoBrokenWeekend*).

Les contraintes ci-dessus sont appliquées à toutes les infirmières. Les exceptions ci-dessous rendraient le problème insoluble :

- Pour les infirmières (830043), (344234) et (841036) l'activation de la contrainte *NoIsolatedShift* rend le problème insoluble. En effet, le respect de la charge du travail, des préférences et des contraintes concernant les fins de semaine rend impossible le respect de cette contrainte pour les infirmières citées ci-dessus.
- Pour les infirmières (901020), (830043) et (831054) l'activation de la contrainte *NoIsolatedOffShift* rend le problème insoluble. En effet, le respect de la charge du travail et des contraintes concernant les fins de semaine rend impossible le respect de cette contrainte pour les infirmières citées ci-dessus.

4.1.2 Unité de pédiatrie de l'Hôpital Royal Victoria (CHILD)

4.1.2.1 Informations générales

Tableau 4.6 : Description des informations générales (jeu CHILD)

Nom de fichier	CHILD_Ju01_N41.data(data2)
Effectif	41 infirmières
Horizon	6 semaines
Nombre de périodes	7
Nombre de quarts	5

4.1.2.2 Décomposition d'une journée

Périodes

Tableau 4.7 : Description des périodes (jeu CHILD)

Numéro de période	Début	Fin
1	07 : 15	10 : 00
2	10 : 00	15 : 15
3	15 : 15	17 : 00
4	17 : 00	18 : 00
5	18 : 00	23 : 15
6	23 : 15	01 : 00
7	01 : 00	07 : 15

Quarts

Tableau 4.8 : Description des quarts (jeu CHILD)

Sigle	Périodes couvertes	Type
D	1, 2	Matin
ET	4, 5, 6	Soir
E	3, 4, 5	Soir
N	6, 7	Nuit
DT	2, 3, 4	Matin

4.1.2.3 La demande

Tableau 4.9 : Description de la demande (jeu CHILD)

Jour	Niveaux	Périodes	Min($\underline{v}$)	Cible	Max($\overline{v}$)
Tous	Tous	1	4	10	22
Tous	Tous	2	4	10	22
Tous	Tous	3	1	3	15
Tous	Tous	4	2	5	17
Tous	Tous	5	1	4	16
Tous	Tous	6	2	5	17
Tous	Tous	7	1	5	17

4.1.2.4 Disponibilité

- Trois infirmières ont une semaine de vacances.
- Cinq infirmières ont deux semaines de vacances.
- Deux infirmières ont trois semaines de vacances.
- Une infirmière a un horaire prédéterminé.

- Six infirmières ont un ou deux quarts préaffectés par semaine.
- Les quarts possibles : voir colonne (quarts possibles) du tableau 4.10 ci-dessous.

4.1.2.5 Équilibrage

Tableau 4.10 : Contraintes Équilibrage (jeu CHILD)

Équilibrage (*Matin, Soir, Nuit*)	Nombre de personnes	Quarts possibles
50 %, 50 %, 0 %	23	D, DT, E, ET
50 %, 0 %, 50 %	1	D, DT, N
55 %, 0 %, 45 %	8	D, DT, N
0 %, 100 %, 0 %	6	E, ET
0 %, 0 %, 100 %	3	N

4.1.2.6 Charge de travail

Pour ce jeu de données, la charge de travail est exprimée par période de deux semaines :

- 15 infirmières sont à temps complet, soit environ 80 heures de travail.
- 26 infirmières ont une charge de travail variant entre 24 et 64 heures.

4.1.2.7 Autres contraintes

Comme pour le jeu BC, les contraintes citées dans la section (4.1.1.7) sont appliquées à toutes les infirmières. Cependant, pour l'infirmière (511668) l'activation de la contrainte *NoBrokenWeekend* rend le problème insoluble. En effet, cette contrainte, *NoBrokenWeekend*, est en conflit avec les préaffectations demandées et la charge de travail.

4.1.3 Unité de dialyse de l'Hôpital Royal Victoria (DIA)

4.1.3.1 Informations générales

Tableau 4.11 : Description des informations générales (jeu DIA)

Nom de fichier	DIA_A01_N38.data (data2)
Effectif	38 infirmières
Horizon	6 semaines
Nombre de périodes	5
Nombre de quarts	5

4.1.3.2 Décomposition d'une journée

Périodes

Tableau 4.12 : Description des périodes (jeu DIA)

Numéro de période	Début	Fin
1	07 : 30	11 : 30
2	11 : 30	15 : 30
3	15 : 30	19 : 30
4	19 : 30	23 : 30
5	23 : 30	07 : 30

Quarts

Tableau 4.13 : Description des quarts (jeu DIA)

Sigle	périodes couvertes	Type
D	1,2	Matin
E	3,4	Soir
DH	2,3	Matin
D2	1,2,3	Matin
E2	2,3,4	Soir

4.1.3.3 La demande

Tableau 4.14 : Description de la demande (jeu DIA)

Jour	Périodes	Min($\underline{v}$)	Cible	Max($\bar{v}$)
Tous sauf dimanche	1	5	12	24
Tous sauf dimanche	2	5	12	24
Tous sauf dimanche	3	4	10	22
Tous sauf dimanche	4	3	9	21
NB : Aucune contrainte de quota n'est appliquée à la période 5				

4.1.3.4 Disponibilité

- Onze infirmières ont une semaine de vacances.
- Une infirmière a quatre semaines de vacances.
- Une infirmière a deux semaines de vacances.
- Une infirmière a un horaire prédéterminé.
- Quatre infirmières ont un ou deux quarts préaffectés par semaine.
- Les quarts possibles : voir colonne (quarts possibles) du tableau 4.15 ci-dessous.

4.1.3.5 Équilibrage

Tableau 4.15 : Contraintes Équilibrage (jeu DIA)

Équilibrage (*Matin, Soir, Nuit*)	Nombre de personnes	Quarts possibles
100 %, 0 %, 0 %	2	D, D2, DH
75 %, 25 %, 0 %	3	D, D2, DH, E, E2
60 %, 40 %, 0 %	19	D, D2, DH, E, E2
50 %, 50 %, 0 %	9	D, D2, DH, E, E2
40 %, 60 %, 0 %	2	D, D2, DH, E, E2
0 %, 100 %, 0 %	3	E, E2

4.1.3.6 Charge de travail

Pour ce jeu de données, la charge de travail est exprimée par période de deux semaines :

- 25 infirmières sont à temps complet, environ 80 heures.
- 13 infirmières ont une charge de travail variante entre 24 et 64 heures.

4.1.3.7 Autres contraintes

Comme pour le jeu BC les contraintes citées dans la section (4.1.1.7) sont appliquées à toutes les infirmières. Cependant, pour l'infirmière (7841) l'activation de la contrainte *NoBrokenWeekend* n'est pas possible parce qu'elle engendre un conflit avec les contraintes préaffectations et charge de travail.

4.1.4 Salle d'urgence de l'Hôpital Général de Montréal (ERMGH)

4.1.4.1 Informations générales

Tableau 4.16 : Description des informations générales (jeu ERMGH)

Nom de fichier	ERMGH_2Juin-13Juillet-2002_N41.data (data2)
Effectif	41 infirmières
Horizon	6 semaines
Nombre de périodes	5
Nombre de quarts	4

4.1.4.2 Décomposition d'une journée

Périodes

Tableau 4.17 : Description des périodes (jeu ERMGH)

Numéro de période	Début	Fin
1	07 : 30	12 : 00
2	12 : 00	15 : 30
3	15 : 30	20 : 00
4	20 : 00	23 : 30
5	23 : 30	07 : 30

Quarts

Tableau 4.18 : Description des quarts (jeu ERMGH)

Sigle	Périodes couvertes	Type
D	1,2	Matin
DH	2,3	Matin
E	3,4	Soir
N	5	Nuit

4.1.4.3 La demande

Tableau 4.19 : Description de la demande (jeu ERMGH)

Jour	Périodes	Min($\underline{v}$)	Cible	Max($\bar{v}$)
Tous	1	3	8	16
Tous	2	3	8	16
Tous	3	3	8	16
Tous	4	3	8	16
Tous	5	3	8	16

4.1.4.4 Disponibilité

- Onze infirmières ont une semaine de vacances.
- Cinq infirmières ont deux semaines de vacances.
- Deux infirmières ont trois semaines de vacances.
- Trois infirmières ont leur horaire prédéterminé,
- Dix infirmières ont entre un et dix quarts préaffectés par semaine.
- Les quarts possibles : voir colonne (quarts possibles) du tableau 4.20 ci-dessous.

4.1.4.5 Équilibrage

Tableau 4.20 : Contraintes Équilibrage (jeu ERMGH)

Équilibrage (*Matin, Soir, Nuit*)	Nombre de personnes	Quarts possibles
100 %, 0 %, 0 %	4	D, DH
60 %, 20 %, 20 %	1	D, DH, E, N
50 %, 50 %, 0 %	1	D, DH, E
50 %, 25 %, 25 %	23	D, DH, E, N
50 %, 0 %, 50 %	4	D, DH, N
0 %, 80 %, 20 %	3	E, N
0 %, 50 %, 50 %	2	E, N
0 %, 0 %, 100 %	3	N

4.1.4.6 Charge de travail

Pour ce jeu de données, la charge de travail est exprimée par période de deux semaines :

- 11 infirmières sont à temps complet, soit environ 80 heures de travail.
- 30 infirmières ont une charge de travail variant entre 24 et 64 heures.

4.1.4.7 Autres contraintes

Comme pour le jeu BC, les contraintes citées dans la section (4.1.1.7) sont appliquées à toutes les infirmières. Cependant, pour l'infirmière (7019) l'activation de la contrainte *NoBrokenWeekend* rend le problème insoluble. En effet, le respect des préaffectations rend impossible l'activation de la contrainte *NoBrokenWeekend.* Aussi, pour l'infirmière (6676) le nombre maximum de quarts consécutifs est de neuf au lieu de sept dû aux préaffectations demandées.

4.2 Plan d'expérimentation

Dans cette section, nous commençons par identifier les jeux utilisés pour la validation des heuristiques proposées. Puis, nous présentons les stratégies retenues pour les tests et le protocole expérimental suivi. Finalement, nous présentons les résultats obtenus lors des tests finaux.

4.2.1 Jeux et paramètre α

Comme indiqué dans la section (4.1), nous avons retenu pour les tests quatre jeux de données de base (BC, DIA, ERMGH et CHILD). Afin d'enrichir nos jeux de données, nous avons généré huit jeux distincts : deux exemplaires de chaque jeu de base. Les deux exemplaires de chaque jeu de base sont identiques, sauf au niveau du paramètre *DENShiftsRatioPrecision*.

Le paramètre *DENShiftsRatioPrecision* permet de déterminer la marge de manœuvre permise pour les contraintes Équilibrage. Par exemple, pour une contrainte Équilibrage spécifiant que :

- 50 % des quarts travaillés doivent être du *Matin*,
- 50 % restants doivent être de *Soir*.
- Si le *DENShiftsRatioPrecision* est égal à 0.15 (15 %) alors les bornes seront :

$\underline{v}_{Matin} = Max\,(50-(0.15*100)\,,\,0)=35\%$ *et* $\bar{v}_{Matin} = Min\,(\,50+(0.15*100)\,,\,100)=65\%$

$\underline{v}_{Soir} = Max\,(50-(0.15*100)\,,\,0)=35\%$ *et* $\bar{v}_{Soir} = Min\,(50+(0.15*100)\,,\,100)=65\%$

$\underline{v}_{Nuit} = Max\,(0-(0.15*100)\,,\,0)=0\%$ *et* $\bar{v}_{Nuit} = Min\,(0+(0.15*100)\,,\,100)=15\%$

Autrement dit, la contrainte impose qu'entre 35 % et 65 % des quarts travaillés doivent être de type *Matin* et la même chose pour les quarts de type *Soir*, alors qu'entre 0 % et 15 % des quarts travaillés doivent être de type *Nuit*. Le tableau 4.21 donne la liste des jeux utilisés et la valeur du paramètre *DENShiftsRatioPrecision* pour chaque jeu.

Tableau 4.21 : Jeux de données retenus.

Jeu de Base	BC	BC	CHILD	CHILD	DIA	DIA	ERMGH	ERMGH
Jeu	BC10	BC15	CHILD10	CHILD15	DIA10	DIA15	ERMGH10	ERMGH15
DENShiftsRatioPrecision	10 %	15 %	10 %	15 %	10 %	15 %	10 %	15 %

Réglage du paramètre α :

Rappelons que ce paramètre définit la tendance d'instanciation des variables durant le déroulement de la recherche. Des tests préliminaires (annexe 3) nous ont permis de fixer ce paramètre à $\frac{1}{2}$.

4.2.2 Stratégies retenues

Dans Hibiscus, trois choix déterminent la stratégie de recherche : la décomposition, le choix de la variable et le choix de la valeur. Cette section traite des différentes stratégies de recherche retenues.

a. Décomposition

La décomposition concerne la partition du problème : comme indiqué dans le chapitre 2, on distingue deux décompositions désignées par DBD et NBN. Vu que nous sommes dans un contexte où les contraintes horizontales sont plus nombreuses et plus serrées (salle des soins infirmiers), nous avons retenu la décomposition horaire par horaire, soit infirmière par infirmière (NBN pour *nurse by nurse*). Le choix des infirmières se fera d'une façon lexicographique, c'est-à-dire selon l'ordre de leur apparition dans le jeu de données.

b. Choix de la variable

Dans notre cas, puisque nous utilisons une décomposition NBN, le choix de la variable concerne le choix du jour à instancier. Pour cet ordre nous avons opté pour deux façons : l'ordre lexicographique (selon le calendrier) et l'ordre 'échec en premier' (*First*

Fail). Le dernier choix (*First Fail*) est basé sur la taille des domaines : on choisit le jour ayant la taille de domaine la plus petite.

c. Choix de la valeur

Nous avons conçu quatre heuristiques d'incitation (Demande, Max_Déficit, Indépendante et Pondérée) afin de mieux guider la recherche en effectuant un meilleur choix de la valeur à instancier.

Les tests préliminaires (voir annexe 4) montrent que les heuristiques d'incitation Indépendante et Pondérée n'ont pas réussi à améliorer la robustesse d'Hibiscus. En effet, elles n'étaient même pas capables d'égaler la stratégie aléatoire. Toutefois, il faut souligner que par manque de temps nous n'avons implanté que des versions très simples de ces heuristiques. Donc, les résultats de l'annexe 4 doivent être considérés comme non concluants. Pour la suite des opérations, nous nous concentrons sur les deux heuristiques que nous avons implantées d'une façon complète, soit l'heuristique Demande et l'heuristique Max_Déficit. Pour ce qui suit, l'heuristique d'incitation Équilibrage fait référence à l'heuristique Max_Déficit.

Rappelons que la partition utilisée est infirmière par infirmière (NBN), w_D est le poids (pondération) de l'heuristique Demande et w_E est le poids de l'heuristique Équilibrage.

Parmi les stratégies retenues, nous avons deux stratégies aléatoires, Rand_Chrono (l'ordonnancement des jours : chronologique) Rand_MinDom (Choix des jours : plus petit domaine). Ces stratégies constituent la base de comparaison des différentes stratégies utilisant les heuristiques proposées. Concernant les stratégies utilisant les heuristiques d'incitation nous avons retenu 10 stratégies. Ces stratégies sont obtenues par changement des pondérations des heuristiques Demande et Équilibrage. Le tableau 4.22 donne la liste de toutes les stratégies retenues.

Tableau 4.22 : Description des stratégies retenues.

Stratégie	Choix du jour	Demande	Max_Déficit
Rand_Chrono	Chronologique.	$w_D = 0$	$w_E = 0$
Rand_MinDom	Plus petit domaine.	$w_D = 0$	$w_E = 0$
D1_E0_Chrono	Chronologique.	$w_D = 1$	$w_E = 0$
D1_E0_MinDom	Plus petit domaine.	$w_D = 1$	$w_E = 0$
D0_E1_Chrono	Chronologique.	$w_D = 0$	$w_E = 1$
D0_E1_MinDom	Plus petit domaine.	$w_D = 0$	$w_E = 1$
D1_E1.5_Chrono	Chronologique.	$w_D = 1$	$w_E = 1.5$
D1_E1.5_MinDom	Plus petit domaine.	$w_D = 1$	$w_E = 1.5$
D1_E1_Chrono	Chronologique.	$w_D = 1$	$w_E = 1$
D1_E1_MinDom	Plus petit domaine.	$w_D = 1$	$w_E = 1$
D1_E0.5_Chrono	Chronologique.	$w_D = 1$	$w_E = 0.5$
D1_E0.5_MinDom	Plus petit domaine.	$w_D = 1$	$w_E = 0.5$

Les stratégies D1_E0_Chrono et D1_E0_MinDom utilisent l'heuristique demande seule, alors que les stratégies D0_E1_Chrono et D0_E1_MinDom utilisent l'heuristique Équilibrage seule. Les autres stratégies utilisent des combinaisons des deux heuristiques. Rappelons qu'une affectation a_i donnée peut être incité simultanément par plusieurs heuristiques Demande. Mais, elle est incitée au plus par une heuristique Max_Déficit. Donc, dans les stratégies D1_E0.5_Chrono et D1_E0.5_MinDom l'heuristique Équilibrage permet d'effectuer le bris d'égalité entre affectations concurrentes ayant obtenu un même score d'incitation par les heuristiques Demande.

4.2.3 Protocole expérimental

Les tests effectués visent à mesurer l'apport des heuristiques d'incitation à la programmation par contraintes. L'apport spécifique visé est l'amélioration de la robustesse d'Hibiscus. Cette amélioration de la robustesse est concrétisée par le maintien des performances à un bon niveau malgré les perturbations. On entend par perturbations les changements de contextes obtenus par l'utilisation de plusieurs jeux de données.

Le système Hibiscus est programmé en langage C++ et utilise la bibliothèque de programmation par contraintes Ilog Solver v5.1. Les tests sont réalisés sur une machine Sun Ultra-10 à 1.2 GHz avec 1Go de mémoire vive sous Sun Solaris 2.9.

Le bris d'égalité, le départage des couples (variable, valeur) ayant un même score d'incitation, s'effectue d'une façon aléatoire (distribution uniforme) à l'aide d'un

générateur pseudo aléatoire. Notons que les bris d'égalité ne surviennent que rarement (moins de 1 %). Nous avons effectué 100 exécutions de chaque jeu de données avec chaque stratégie retenue (tableau 4.22). Les germes aléatoires sont différents d'une exécution à une autre. Le temps d'exécution maximum d'une exécution est 500 secondes (voir annexe 5), une exécution est considérée comme un échec si aucune solution n'est trouvée durant le temps d'exécution maximum.

Nous effectuons le suivi du pourcentage de succès (# de succès/# d'exécutions), du temps moyen d'exécution des succès et de l'écart type des temps d'exécution. Le pourcentage de succès est notre paramètre principal.

4.3 Résultats

Dans cette section, nous présentons un résumé des résultats obtenus. Trois mesures de performance seront présentées : le pourcentage de réussite, le temps moyen d'exécution et l'écart type des temps d'exécution.

Nous n'effectuons aucune comparaison concernant la qualité des solutions obtenues. Rappelons que nous sommes dans un contexte de satisfaction de contraintes et non d'optimisation. Toutefois, Hibiscus garantit que les solutions obtenues sont de bonne qualité vis-à-vis des nombreux critères pris en compte (les contraintes considérées).

4.3.1 Pourcentage de réussite

Dans cette section, nous présentons les pourcentages de réussite, le tableau (4.24) et la figure (4.1). Nous comparons la robustesse des stratégies : tout d'abord au niveau de leur capacité de résolution des jeux retenus, ensuite au niveau des pourcentages de réussite.

Afin de mieux visualiser les résultats obtenus et faciliter leur lecture, nous utilisons le code de couleurs indiqué dans le tableau (4.23).

Tableau 4.23 : Code de couleurs pour les pourcentages de réussite

Marge (%)	0 - 9	10 - 24	25 - 49	50 - 74	75 - 100
Couleur	Rouge	Violet	Jaune	Bleu	Vert

Premièrement, on peut noter que le jeu BC, dans ses deux versions, semble être le jeu le plus facile. En effet, toutes les stratégies ont pu obtenir un pourcentage de réussite supérieur à 50 % dans la résolution de ce dernier. Nous pouvons aussi constater que la plupart des stratégies arrivent souvent à résoudre les jeux de données retenus. La seule exception se situe au niveau des stratégies aléatoires (Rand_Chrono, Rand_MinDom) et les stratégies utilisant seulement l'heuristique Équilibrage (D0_E1_Chrono et D0_E1_MinDom) à la résolution des jeux CHILD_10 et CHILD_15.

Tableau 4.24 : Résultats pourcentage de réussite

	BC10	BC15	DIA10	DIA15	ERMGH10	ERMGH15	CHILD10	CHILD15
Rand_Chrono	54 %	58 %						
Rand_MinDom	55 %	52 %						
D1_E0_Chrono	67 %	70 %	14 %	27 %				
D1_E0_MinDom	61 %	58 %	24 %	31 %				
D0_E1_Chrono	66 %	63 %						
D0_E1_MinDom	55 %	53 %						
D1_E1.5_Chrono	79 %	81 %	14 %	41 %		27 %	22	
D1_E1.5_MinDom	57 %	55 %		36 %				
D1_E1_Chrono	83 %	85 %	29 %	47 %	16 %	31 %	22	26 %
D1_E1_MinDom	52 %	64 %		41 %		27 %		
D1_E0.5_Chrono	78 %	82 %		43 %		28 %		28 %
D1_E0.5_MinDom	56 %	59 %		31 %				

Les stratégies aléatoires, stratégie de base : Rand_Chrono et Rand_MinDom, sont généralement peu performantes. En effet, elles n'obtiennent un bon pourcentage qu'avec les jeux BC10 et BC15, soit les jeux les plus faciles, alors que pour les autres jeux le pourcentage de réussite reste en dessous de 10 %. Les stratégies D0_E1_Chrono et D0_E1_MinDom, soit celles utilisant l'heuristique Équilibrage seule, n'ont pas pu faire mieux que les stratégies aléatoires (stratégies de base).

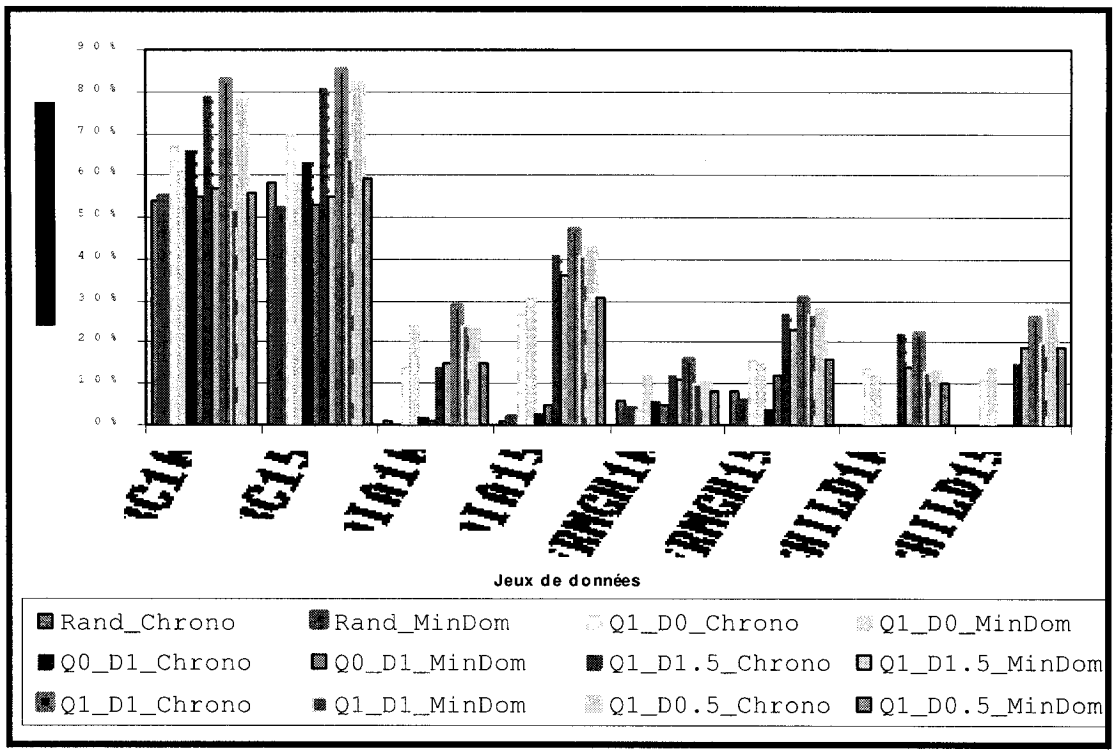

Figure 4.1 : Graphe pourcentage de réussite

Généralement, les stratégies D1_E0_Chrono et D1_E0_MinDom, soit celles utilisant l'heuristique Demande seule, sont plus robustes que les stratégies aléatoires pour tous les jeux de données. En effet, les pourcentages de réussite obtenus avec les stratégies D1_E0_Chrono et D1_E0_MinDom sont plus importants que ceux obtenus avec les stratégies aléatoires (stratégies de base).

Regardons maintenant ce qui se passe lors de l'utilisation des deux heuristiques d'incitation, soit l'heuristique Demande et l'heuristique Max_Déficit. Généralement, les stratégies utilisant ces heuristiques sont les plus performantes; on note une amélioration intéressante de la robustesse. L'amélioration de la robustesse est mieux visible pour les

jeux DIA, CHILD et ERMGH, soit ceux pour lesquels la stratégie aléatoire a complètement échouée. Toutefois, aucune stratégie ne domine grandement les autres. Cependant, la stratégie D1_E1_Chrono, pondérations des heuristiques Demande et Équilibrage égales, a obtenu souvent une performance légèrement meilleure que les autres.

Finalement, on note que globalement les stratégies utilisant un ordonnancement des jours chronologique (*_*_Chrono) performent mieux que celles utilisant l'ordonnancement plus petit domaine (*_*_MinDom).

4.3.2 Temps d'exécution moyen et écart type des temps d'exécution

Dans cette section, nous présentons les résultats concernant les temps moyens obtenus, tableau (4.26). Afin de mieux visualiser les résultats obtenus et faciliter leur lecture, nous utilisons le code de couleurs du tableau (4.25). Chaque case du tableau (4.26) indique le temps d'exécution moyen et l'écart type. Les chiffres entre parenthèses sont les écart type obtenus.

Tableau 4.25 : Code de couleurs pour les pourcentages de réussite

Marge temps d' exécution (sec)	0 - 59	60 - 119	120 - 179	> 180
Couleur	Vert	Bleu	Jaune	

NB : seules les stratégies ayant obtenu des pourcentages élevés pour tous les jeux sont retenues pour l'analyse des temps moyens d'exécution et de son écart type.

Tableau 4.26 : Résultats : Temps moyen d'exécution (secondes)

	BC10	BC15	DIA10	DIA15	ERMGH10	ERMGH15	CHILD10	CHILD15
D1_E1.5_Chrono	119 (93)	145 (101)	100 (70)	90 (100)	163 (125)	59 (67)	94 (56)	115 (139)
D1_E1.5_MinDom	89 (68)	105 (67)	169 (109)	75 (96)		70 (96)	110 (45)	75 (71)
D1_E1_Chrono	112 (73)	156 (110)	68 (53)	64 (88)	178 (159)	101 (87)	131 (133)	91 (91)
D1_E1_MinDom	97 (62)	117 (77)	101 (106)	68 (100)	88 (46)	99 (99)	110 (89)	111 (96)
D1_E0.5_Chrono	127 (104)	129 (88)	128 (68)	60 (70)	159 (156)	76 (96)	94 (76)	88 (110)
D1_E0.5_MinDom	95 (77)	124 (85)	64 (46)	52 (78)	130 (92)	65 (167)	125 (119)	88 (66)

Tout d'abord, il faut préciser que le temps d'exécution ne constitue pas un critère principal dans ce travail, puisque notre objectif capital est l'amélioration de la robustesse.

Généralement, on peut noter que les temps d'exécution moyens sont comparables. En effet, la majorité des temps d'exécution sont situés dans la zone 1 à 3 minutes (couleurs : bleu et jaune tableau 4.26). On note aussi que les performances, temps moyen et écart type, de la meilleure stratégie (D1_E1_Chrono) se situent dans la moyenne.

On constate que la mauvaise performance (cases rouges) concerne le jeu ERMGH10. Ce dernier est le jeu le plus difficile, d'ailleurs les pires pourcentages sont enregistrés aussi pour ce dernier.

Concernant les écarts types, on voit clairement qu'aucune stratégie ne se distingue des autres. En effet, la majorité des écarts types sont entre 60 et 120. Comme pour les deux performances précédentes, pourcentages de réussite et les temps d'exécution moyens, on constate que les écarts types les plus grands (mauvaises performances) sont obtenus avec le jeu le plus difficile (ERMGH10).

4.4 Discussion

Le choix du temps maximal a probablement biaisé les résultats au niveau du temps moyen et de l'écart type. Mais vu que ces deux critères ne sont pas principaux, on a gardé ce choix puisqu'il ne biaise pas le critère principal, la robustesse.

On prend comme base de comparaison la stratégie aléatoire (Rand_Chrono). L'utilisation de l'heuristique Équilibrage toute seule (D0_E1_Chrono) n'amène pratiquement aucune amélioration de la robustesse. Cependant, l'utilisation de l'heuristique Demande seule permet une bonne amélioration de la robustesse surtout avec les jeux DIA et CHILD où la stratégie aléatoire a complètement échoué.

L'utilisation des deux heuristiques (D1_E1.5_Chrono, D1_E1_Chrono et D1_E0.5_Chrono) améliore significativement la robustesse du système. Généralement, toutes les stratégies utilisant les deux heuristiques, Demande et Max_Déficit, ont obtenu des performances comparables.

En comparant les stratégies utilisant les heuristiques et les stratégies aléatoires, en faisant abstraction des stratégies D0_E1_Chrono et D0_E1_MinDom, on remarque la supériorité de celles utilisant les heuristiques d'incitation. Généralement les stratégies Rand_Chrono, Rand_MinDom, D0_E1_Chrono et D0_E1_MinDom sont celles qui ont obtenu les plus mauvaises performances.

En ce qui concerne le choix de variable (le jour), on rappelle que globalement les stratégies utilisant un ordonnancement chronologique (*_*_Chrono) performent mieux que celles utilisant l'ordonnancement 'le plus petit domaine d'abord' (*_*_MinDom). L'ordonnancement chronologique obtient les meilleures performances probablement parce que cet ordonnancement permet la satisfaction des contraintes horizontales, principalement les contraintes ergonomiques, plus facilement. Par exemple, une contrainte de succession des quarts sera probablement plus facile à satisfaire si les jours sont instanciés chronologiquement. Rappelons que les contraintes horizontales sont les plus nombreuses dans les problèmes de confection d'horaires.

Finalement, la stratégie la plus robuste, figure 4.1, reste la stratégie D1_E1_Chrono. Rappelons que la stratégie D1_E1_Chrono utilise une partition NBN (infirmière par infirmière), le choix de la variable (le jour) est chronologique et que le choix de la valeur s'effectue à l'aide des deux heuristiques d'incitation Demande et Équilibrage avec les pondérations suivantes :

- $w_{h_D} = 1$ poids de l'heuristique Demande,
- $w_{h_{éq}} = 1$ poids de l'heuristique Équilibrage.

Conclusion

Dans ce mémoire, nous avons proposé un modèle de conception d'heuristiques d'incitation. À partir du modèle proposé, nous avons conçu deux heuristiques d'incitation. La première heuristique concerne les contraintes de type Cardinalité, alors que la deuxième concerne les contraintes de type Cardinalité pondéré.

Les deux heuristiques proposées sont très génériques. D'ailleurs, leurs adaptations aux contraintes retenues, Demande et Équilibrage, sont quasi directes. Notons que les contraintes retenues sont largement répandues dans les problèmes de confection d'horaire et elles représentent souvent un défi intéressant.

Nous avons testé la robustesse des heuristiques proposées en les utilisant comme heuristiques de choix de la valeur dans la stratégie de recherche d'une application complexe en programmation par contraintes (Hibiscus). Les heuristiques d'incitation conçues peuvent aussi être utilisées comme heuristiques de choix de la variable. Notons aussi que les heuristiques proposées peuvent être facilement utilisées dans les stratégies de recherche (comme heuristique de choix de la valeur ou de la variable) avec des paradigmes de programmation autre que la programmation par contraintes.

Nous avons utilisé plusieurs contextes réels et hétérogènes pour vérifier l'impact de l'intégration des heuristiques d'incitation sur la robustesse du système. Nous considérons que les résultats obtenus sont très encourageants. Toutefois, nos conclusions sont à tempérer par le fait que notre base de test reste relativement réduite. Cependant, les

données sont suffisamment hétérogènes pour montrer que les heuristiques d'incitation ont amélioré significativement la robustesse d'Hibiscus. On peut aussi déplorer le manque de références de comparaison avec d'autres méthodes sur les mêmes données (benchmark externe). Un benchmark externe aurait permis d'identifier clairement le niveau de difficultés des jeux de données utilisées.

Au terme de ce travail, Hibiscus se révèle être un outil intéressant. Il est flexible : s'adapte parfaitement à plusieurs contextes. Aussi, il permet de définir des stratégies adaptées et flexibles. Cependant, Hibiscus n'est pas encore utilisable en pratique, car il ne permet pas de modéliser de façon systématique les contraintes souples comme l'équité.

Ci-dessous, quelques pistes de développement que nous jugeons intéressantes :

- Étendre la conception des heuristiques d'incitation aux autres types de contraintes.
- Réétudier les heuristiques d'incitation Pondérée et Indépendante. Quoique les résultats obtenus avec ces heuristiques étaient médiocres, nous pensons que ces deux heuristiques peuvent s'avérer importantes. Rappelons que nous avons implémenté des versions simples de ces heuristiques.
- Pendant les tests préliminaires, nous avons testé différents ordonnancements d'infirmières. Les résultats ont montré quelques améliorations du pourcentage de réussite.
- Vu que les temps d'exécution sont faibles (inférieure à 10 minutes) une stratégie basée sur la relance, selon un certain schéma, peut s'avérer une bonne stratégie de recherche.
- Le benchmark externe, même avec d'autres paradigmes de programmation (mathématiques, et programmation linéaire, ...), permettra de mieux situer les capacités d'Hibiscus et d'orienter les travaux de recherche futurs.

Bibliographie

[1] Stéphane Bourdais. Génération automatique d'horaires en milieu hospitalier. Mémoire de maître ès sciences appliquées (M.Sc.A.), Université de Montréal, Département génie informatique, École Polytechnique, Septembre 2003.

[2] Stéphane Bourdais, Philippe Galinier, Gilles Pesant. *HIBISCUS : A Constraint Programming Application to Staff Scheduling in Health Care*, in Rossi, F. (ed.), Proceedings, 9th Principles and Practice of Constraint Programming (CP 2003), 2003, 153-167

[3] Gilles Pesant. *Counting Solutions of CSPs: A Structural Approach.* 2005. second International Conference, CPAIRO 2005 Prague, Czech Republic, May 30-June 1 2005. pages : 110-117.

[4] Nicolas Prcovic, Bertrand Neveu. *Progressive Focusing Search*, ECAI 2002, pages : 53-57.

[5] Philippe Refalo. *Impact-Based Search Strategies for Constraint Programming.* CP 2004: 557-571.

[6] Khaled Elbassioni, Irit Katriel. *Multiconsistency and Robustness with Global Constraint.* CPAIOR 2005: 168-182

[7] S. Abdennadher et H. Schenker. *Nurse scheduling using constraint logic programming.* IAAI-99, Orlando, Florida, 1999.

[8] H. Beaulieu. Planification de l'horaire des médecins dans une salle d'urgence. Mémoire de maîtrise, Université de Montréal, 1998.

[9] Buzòn. La confection des horaires de travail des médecins d'urgence résolue à l'aide de la recherche tabou. Mémoire de maîtrise, Université de Montréal, 2001.

[10] A. Caprara, M. Monaci, et P. Toth. Models and algorithms for a staff scheduling problem. Mathematical Programming, 2003.

[11] Y. Caseau, P. Giullo, et E. Levenez. *A deductive and object-oriented approach to a complex scheduling problem. In Third International Conference*, DOOD'93, Phoenix, Arizona, USA, *December* 6-8, 1993.

[12] B. M. W. Cheng, J. H. M. Lee, et J. C. K.Wu. *A constraint-based nurse rostering system using a redundant modeling approach. In 8th IEEE International Conference on Tools with Artificial Intelligence*, IEEE Computer Society Press, 1996.

[13] S. J. Darmoni, A. Fajner, N. Mah_e, A. Leforestier, M. Vondracek, O. Stelian, et M. Baldenweck. *Horoplan : computer-assisted nurse scheduling using contraintbased programming. Journal of the Society for Health Systems*, 1995.

[14] W. Feuilletin. Génération automatique d'horaires d'infirmières à l'aide de la programmation par contraintes. CRT, Canada, Octobre 2000.

[15] F. Forget. Confection automatisée des horaires de médecins dans une salle d'urgence. Mémoire de maîtrise, Université de Montréal, 2002.

[16] K. Heus. Gestion des plannings infirmiers, Application des techniques de programmation par contraintes. Thèse de doctorat, Université Joseph Fourier -Grenoble 1, Mai 1996.

[17] B. Jaumard, F. Semet, et T. Vovor. *A generalized linear programming model for nurse scheduling. European Journal of Operational Research*, 1998.

[18] A. Meisels, E. Gudes, et G. Solotorevsky. *Combining rules and constraints for employee timetabling. International Journal of Intelligent Systems*, 12:419-439, 1997.

[19] H. Meyer auf' m Hofe. *Solving rostering tasks by generic methods for constraint optimization. International Journal of Foundations of Computer Science*, 2001.

[20] L. M. Rousseau, G. Pesant, et M. Gendreau. *A hybrid algorithm to solve a physician rostering problem. In Second Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.* Paderborn, Germany, 2000.

[21] M. Saadie. Planification de l'horaire des médecins dans une salle d'urgence par la programmation par contraintes. Mémoire de maîtrise, Université de Montréal, 2003.

[22] G. Thompson. *A simulated-annealing heuristic for shift scheduling using non-continuously available employees. Computer Operational Research*, 1996.

[23] E. Burke, G. Kendall, E. Soubeiga, E. Costa, J. Martin-Blazquez, P. Ross. *comparison between a HH based local search and based constructive heuristic : A case for hybridisation?.*2001.

[24] E. Burke, G. Kendall, and E. Soubeiga. *A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics*. 2003.

[25] P. Ross, S. Schulenburg, J. G. Martin-Blazquez, and E. Hart. *Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In Proceedings of the Genetic and Evolutionary Computation. Conference,* GECCO'02, 2002.

[26] Patrick Prosser. *The Dynamics of Dynamic Variable Ordering Heuristics.* on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.

[27] Griesmer, H., 1993. Self-scheduling turned us into a winning team. Management Decisions 56 (12), 21–23. Howell, JP, 1998.

[28] Dechter, R. and Pearl. *Network-Based Heuristics for Constraint-Satisfaction Problems.* Artificial Intelligence 1988, 34-38.

[29] www.*en.wikipedia.org*

Période de consultation : Septembre Décembre 2004

[30] http://rfv.insa-lyon.fr/~jolion/STAT/node68.html

Période de consultation : Septembre Décembre 2004

[31] www.cours.polymtl.ca/inf6101

Période de consultation : Septembre Décembre 2004

[32] *www.cours.polymtl.ca/inf4705* Période de consultation : Septembre Décembre 2004

[33] Note de cours INF6953A, Mr Philippe Galinier. Ecole polytechnique de Montréal Hiver 2005.

[34] Minton et al. Minimizing conflict: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence, 581992.

[35] Daniel Frost et Rina Dechter. *In search of the best constraint satisfaction search.* Proceedings of the National Conference on Artificial Intelligence, Seattle, WA, August 1994, pp. 301--306.

[36] Burke EK, De Causmaecker P, Vanden Berghe G and Van Landeghem H (2004) The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7(6): 441-499.

[37] *http://www.tout-savoir.net*

Période de consultation : Septembre Décembre 2004

[38] Schedule Pro. Logiciel commerciale de confection d'horaire.

http://www.edpsoftware.com/

Période de consultation : Septembre Décembre 2004

[39] *Horoplan: computer-assisted nurse scheduling using constraint-based programming, http://www.chu-rouen.fr/dsii/publi/plao.html*

Période de consultation : Septembre Décembre 2004

[40] COSYTEC. Gymnaste.

http ://www.cosytec.com/constraint-programming/casesstudies/health-care.htm.

Période de consultation : Septembre Décembre 2004

[41] *Interactive Information R&D. E_ective Computer Aided Scheduling.*

http ://www.i2rd.com/ECAS,

Période de consultation : Septembre Décembre 2004

[42] InTime. SpeedShift. *http ://www.intimesoft.com,*

Période de consultation : Septembre Décembre 2004

[43] D. Fost and R. Dechter. *Look-ahead value ordering for constraint satisfaction problems. In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[44] J. Larrosa and P. Meseguer. *Optimization-based Heuristics for Maximal Constraint Satisfaction. In Principles and Practice of Constraint Programming* CP'95, 1995.

[45] Norman Sadeh and Mark S. Fox. *Variable and Value Ordering Heuristics for Activity-based Jobshop Scheduling. In Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, 1990.

[46] *http://www.cosytec.fr/ordonnancement_de_production/chip/technologie_chip.htm*

Période de consultation : Septembre Décembre 2004

[47] *http://www.icparc.ic.ac.uk/eclipse/*

Période de consultation : Septembre Décembre 2004

[48] *http://pauillac.inria.fr/~diaz/gnu-prolog/*

Période de consultation : Septembre Décembre 2004

[49] *http://www.mozart-oz.org/* Période de consultation : Septembre Décembre 2004

[50] *http://choco.sourceforge.net/*

Période de consultation : Septembre Décembre 2004

[51] *http://www.ilog.fr/products/*

Période de consultation : Septembre Décembre 2004

[52] Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. In Proceedings of the 14th National Conference on AI, pages 327{333. American Association for Articial Intelligence, 1997.

[53] D. Frost, R. Dechter, Look-ahead value ordering for constraint satisfaction problems, in: Proc. IJCAI-95, Montreal. Quebec, 1995, pp. 572-578.

[54] W. Feuilletin. G_en_eration automatique d'horaires d'in_rmi_eres _a l'aide de la programmation par contraintes. Rapport Technique C7PQMR PO2000-32-X, Centre de Recherche sur les Transports, Canada, Octobre 2000.

[55] FELISIAK Julien, Brigitte Jaumard et Gilles Pesant, 'HIBIScus : génération d'horaire d'infirmières avec la programmation par contraintes', avril à août 2001.

Annexes

Annexe 1 : Quarts non couverts : Heuristique Demande.

Ci-dessous un exemple montrant le problème d'instanciation tardive des quarts de travail.

On suppose qu'on a 12 infirmières et qu'on a 3 quarts de travail D, L et N. chaque quart à une contrainte Demande comme suit :

D : min 3, max 6.

L : min 3, max 6.

N : min 2, max 4.

Supposant que toutes les variables qui ne sont pas encore instanciées ont des domaines {D, L, N, O}. Avec 'O' le quart off.

Tableau 7.1 : Problème d'instanciation tardive des quarts de travails.

				Action
	D	L	N	
L	6	6	6	D, L et o sont incités par N
D	5.5	6.5	5.5	D et o sont doublement incités par N,L
O	6	6	5	D, L et o sont incités par N 'on choisi off'
O	5.5	5.5	4.5	D, L et o sont incités par N 'on choisi off'
O	5	5	4	Aucun quart n'est incité. 'on choisi off'
[redacted]				
?	3.5	3.5	3	Aucun quart n'est incité. 'on choisi off'
?				

Donc, rendu à la case rouge, on a atteint le maximum de quarts off qu'on peut avoir, mais très peu de quarts de travail. Tous les quarts restants doivent être des quarts de travails donc la marge de manœuvre est serrée. Le problème vient dans ce cas du quart off (dans le cas général, ça sera tous les quarts qui n'ont pas de quota). Le quart off est libre donc il est souvent incité par les autres heuristiques. En effet, chaque fois qu'un quart de travail est incité le quart off l'est aussi.

Application de la solution proposée à l'exemple précédent :

Dans ce cas, on va faire le suivi du quart off aussi. Alors, on aura les contraintes Demande :

- D : min 3, max 6.
- L : min 3, max 6.
- N : min 2, max 4.
- Off : min 0; max 4. (max = 12 – 3 -3 -2; min = 12 -6 -6 -4 = -4 ->0)

Tableau 7.2 : Résolution du problème d'instanciation tardive des quarts de travails.

		Estimation				Action
		D	L	N	O	
L		6	6	6	6	D et L doublement incité par N et O.
D		5.5	6.5	5.5	5.5	D triplement incité par L, N et O. N et O incité par L. L doublement incité par N et O.
L		6	6	5	5	D et L doublement incité par N et O.
D		5.5	6.5	4.5	4.5	D triplement incité par L, N et O. N et O incité par L. L doublement incité par N et O.
O		6	6	4	4	Personne n'est incité.
N		5.5	5.5	3.5	4.5	D, L et N sont incités par O.
O		5	5	4	4	Personne n'est incité.
N		4.5	4.5	3.5	4.5	D, L et N sont incités par O.
O		4	4	4	4	Personne n'est incité.
N		3.5	3.5	3.5	4.5	D, L et N sont incités par O.
D		3	3	4	4	D et L sont incités par leurs heuristiques
L		3.5	2.5	3.5	3.5	L est incité par son heuristique

Dans ce cas, on a une meilleure distribution des quarts de travail et de repos.

Annexe 2 : Contraintes supportées par Hibiscus (Infirmières)

- *FeasibleQuarts* : les quarts qu'un membre du personnel peut effectuer.
- *PreAssignments* : Affectations prédéterminées.
- *ForbiddenAssignments* : Affectations interdites.
- *NoWEVacations* : Pas de vacances les fins de semaines.
- *Vacations* : Les vacances.
- *WEVacSoft* : Si le vendredi et le lundi suivant ne sont pas travaillés. Alors, la fin de semaine située entre les deux ne doit pas être travaillée.
- *WEVacHard* : Si le vendredi n'est pas travaillé alors, le samedi suivant ne doit pas être travaillé. Le dimanche ne doit pas être travaillé si le lundi suivant ne l'est pas.
- *Quota* : La demande exprimée par jour par niveau de compétence par période.
- *WorkLoad* : Charge de travail du personnel exprimé en nombre d'heures par intervalle de deux semaines.
- *DENQuartsRatio* : Définis des densités <min et max> par type de quart par membre.
- *ConsecutiveWeekends* : définis la succession des fins de semaines travaillées et non travaillées.
- *NoBrokenWeekend* : On ne brise pas les fins de semaines.
- *NextDayShift* : Succession des quarts.
- *ConsecutiveShift* : langueurs maximales des quarts consécutifs de même type.
- *ConsecutiveWorkShift* : langueur maximale des quarts de travail consécutif.
- *DENQuartsRatioPrecision* : Facteur de précision indiquant l'écart maximal valide permis pour les ratios *DENQuartsRatio* des jours, des soirs et des nuits.

Annexe 3 : Détermination paramètre α

Nous avons effectué des tests préliminaires pour déterminer la valeur du paramètre α. Nous avons lancé 50 exécutions. Les jeux utilisés sont BC, CHILD, DIA et ERMGH de base avec les contraintes Équilibrage désactivées. La stratégie utilisée est D1_E0_Chrono et le temps maximal d'exécution est de 100 secondes.

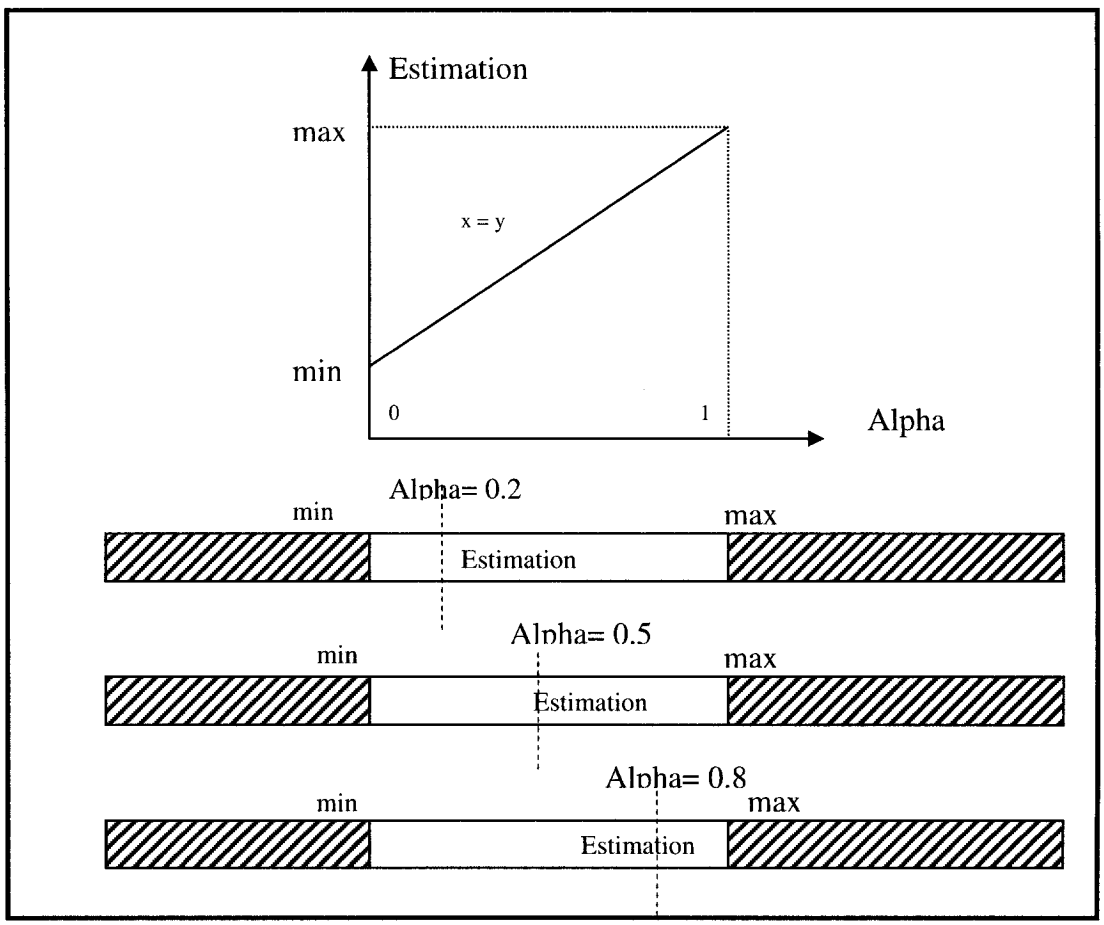

Figure 7.1 : paramètre Alpha

Rappelons que le paramètre Alpha définit la tendance d'instanciation.

- Alpha petit : on est proactif, inquiet, on effectue des incitations très tôt.
- Alpha grand : on laisse aller un peu, optimiste, on retarde les incitations.

Tableau des résultats

Tableau 7.3 : Pourcentage de réussite en fonction de α.

Alpha	BC	CHILD	DIA	ERMGH
0.1	0 %	0 %	0 %	24 %
0.2	0 %	0 %	0 %	30 %
0.3	0 %	4 %	0 %	6 %
0.4	0 %	2 %	6 %	18 %
0.5	80 %	36 %	28 %	48 %
0.6	80 %	0 %	8 %	0 %
0.7	0 %	0 %	0 %	0 %
0.8	0 %	0 %	0 %	0 %
0.9	0 %	0 %	0 %	0 %

Généralement, les meilleures performances ont étaient obtenu au alentour de $\alpha = 0.5$ ($\alpha = 0.5$ veut dire qu'on n'est ni proactif ni insouciant, on garde la tendance). Ce pendant, le jeu ERMGH à un comportement différent. En effet, même avec des Alpha = 0.1 ou 0.2 nous avons obtenus des résultats intéressants pour certains jeux. Concernant les temps d'exécutions, les performances sont en général comparables.

Annexe 4 : Heuristiques d'incitation Indépendante et Ponderée

Protocole de test :

Jeux :

Huit exemplaires de jeux sont considérés :

Tableau 7.4 : Exemplaires des jeux retenus

Ratio de précision	BC	CHILD	DIA	ERMGH
15 %	BC10	CHILD 10	DIA10	ERMGH10
10 %	BC15	CHILD 15	DIA15	ERMGH15

Heuristiques utilisées :

- H_{Dem} : Heuristique de Cardinalité (Demande).
- H_{dis_Ind} : Heuristique Indépendante, le suivi de chaque type (J, S, N) est géré par une heuristique indépendamment des autres types. Tous les types (J, S, N) ont le même poids.
- H_{dis_Pond} : Heuristique Pondérée, le suivi de chaque type (J, S, N) est géré par une heuristique indépendamment des autres. Sauf qu'ici le poids de chaque type dépend de sa Cardinalité.
- $H_{max\_déficit}$: Heuristique d'Équilibrage Max_Déficit, le suivi des types (J, S, N) est géré par une seule heuristique. Seul le type le plus déficient est incité.

Nombre d'exécution et temps max :

Vingt exécutions sont lancées avec un temps d'exécution max de 1000 secondes. Si après 1000 secondes aucune solution n'est trouvée, l'exécution est considérée comme un échec.

Résumé des résultats :

Tableau 7.5 : Pourcentage de réussite heuristiques Indépendante et Pondérée

	BC10	BC15	DIA10	DIA15	ERMGH10	ERMGH15	CHILD10	CHILD15
H_{Dem}	65 %	70 %	15 %	25 %	5 %	15 %	15 %	10 %
H_{Dem} + H_{dis_Ind}	40 %	45 %	10 %	10 %	0 %	0 %	0 %	0 %
H_{Dem} + H_{dis_Pond}	30 %	45 %	10 %	10 %	0 %	0 %	0 %	0 %
H_{Dem} + $H_{Max\_Déficit}$	75 %	80 %	25 %	40 %	10 %	30 %	15 %	25 %

On remarque que l'ajout des heuristiques Indépendante et Pondérée a dégradé la robustesse du système. Alors que l'ajout de l'heuristique Max Déficit a amélioré sensiblement la robustesse d'Hibiscus.

Annexe 5 : Détermination Temps max d'exécution

Cette section présente les tests préliminaires que nous avons effectués pour déterminer le temps d'exécution max à préconiser lors des tests finaux.

Stratégie : D1_E1_Chrono

Temps max d'exécution : 200 secondes.

Nombre de tests : 50.

Jeux de données : les BC10, BC15, DIA10, DIA15, ERMGH10, ERMGH15, CHILD10 et CHILD15.

Le tableau des résultats (page suivante) est trié dans l'ordre croissant afin de mieux visualiser l'ordre de grandeur des temps d'exécution obtenus. On remarque qu'au-delà de 90 % des tests réussis ont terminé dans un temps inférieur à 500 secondes. Donc, pour les tests finaux nous préconisons un temps maximum d'exécution de 500 secondes. Ci-dessous les résultats obtenus.

Tableau 7.6 : Détermination temps maximal d'exécution

#	BC10	BC15	DIA10	DIA15	ERMGH10	ERMGH15	CHILD10	CHILD15
8	70	21	294	32	209	60	229	65
9	75	37	325	41		78	417	68
10	75	38	382	42		237	442	74
11	81	40		49		332		92
12	83	52		51		413		106
13	87	72		55				160
14	88	85		67				375
15	88	96		69				477
16	91	98		75				
17	94	101		97				
18	116	103		224				
19	118	105		290				
20	137	109		312				
21	144	109						
22	146	128						
23	147	135						
24	148	139						
25	148	148						
26	148	150						
27	150	172						
28	153	173						
29	212	179						
30	213	183						
31	216	183						
32	259	185						
33	275	187						
34	310	219						
35	323	226						
36	336	236						
37	354	264						
38	358	267						
39	394	357						
40	414	365						
41	498	365						
42		409						
43		488						